ACCELERATING DIGITAL FORENSIC SEARCHING THROUGH GPGPU PARALLEL PROCESSING TECHNIQUES

A thesis submitted for the degree of Doctor of Philosophy (PhD)

by

Ethan Bayne

School of Design and Informatics,
Abertay University.

February 2017

Declaration

Candidate's declarations:

I, Ethan Bayne, hereby certify that this thesis submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy (PhD), Abertay University, is wholly my own work unless otherwise referenced or acknowledged. This work has not been submitted for any other qualification at any other academic institution.

Signed ……
Date…….

Supervisor's declaration:

I, Robert Ian Ferguson, hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of Doctor of Philosophy (PhD) in Abertay University and that the candidate is qualified to submit this thesis in application for that degree.

Signed ……
Date…….

Certificate of Approval

I certify that this is a true and accurate version of the thesis approved by the examiners, and that all relevant ordinance regulations have been fulfilled.

Supervisor…….
Date……

Dedication

I would like to thank my supervisors – Dr Robert Ian Ferguson and Dr Adam Sampson – for the countless conversations around the different aspects of this research. Their timely encouragement and suggestions have aided in achieving successes beyond anything we expected at the beginning of this investigation.

A notable mention goes to Dr Lynsay Shepherd and Dr Gavin Hales. Their friendship (and *"bants"*) in the department against the dark arts office has kept me sane for the duration of my PhD studies.

This work is dedicated to my mum and dad for their continued love and support, without it, this research would have been impossible to accomplish. I would also like to thank my wife – Cecilia – for her continued patience and encouragement. Without her emotional support and supply of caffeine at home, I would not have had the same motivation to succeed in my research.

ABSTRACT

Background

String searching within a large corpus of data is a critical component of digital forensic (DF) analysis techniques such as file carving. The continuing increase in capacity of consumer storage devices requires similar improvements to the performance of string searching techniques employed by DF tools used to analyse forensic data.

As string searching is a trivially-parallelisable problem, general purpose graphic processing unit (GPGPU) approaches are a natural fit. Currently, only some of the research in employing GPGPU programming has been transferred to the field of DF, of which, a closed-source GPGPU framework was used— Complete Unified Device Architecture (CUDA). Findings from these earlier studies have found that local storage devices from which forensic data are read present an insurmountable performance bottleneck.

Aim

This research hypothesises that modern storage devices no longer present a performance bottleneck to the currently used processing techniques of the field, and proposes that an open-standards GPGPU framework solution – Open Computing Language (OpenCL) – would be better suited to accelerate file carving with wider compatibility across an array of modern GPGPU hardware. This research further hypothesises that a modern multi-string searching algorithm may be better adapted to fulfil the requirements of DF investigation.

Methods

This research presents a review of existing research and tools used to perform file carving and acknowledges related work within the field. To test the hypothesis, parallel

file carving software was created using C# and OpenCL, employing both a traditional string searching algorithm and a modern multi-string searching algorithm to conduct an analysis of forensic data. A set of case studies that demonstrate and evaluate potential benefits of adopting various methods in conducting string searching on forensic data are given. This research concludes with a final case study which evaluates the performance to perform file carving with the best-proposed string searching solution and compares the result with an existing file carving tool— Foremost.

Results

The results demonstrated from the research establish that utilising the parallelised OpenCL and Parallel Failureless Aho-Corasick (PFAC) algorithm solution demonstrates significantly greater processing improvements from the use of a single, and multiple, GPUs on modern hardware. In comparison to CPU approaches, GPGPU processing models were observed to minimised the amount of time required to search for greater amounts of patterns. Results also showed that employing PFAC also delivers significant performance increases over the BM algorithm. The method employed to read data from storage devices was also seen to have a significant effect on the time required to perform string searching and file carving.

Conclusions

Empirical testing shows that the proposed string searching method is believed to be more efficient than the widely-adopted Boyer-Moore algorithms when applied to string searching and performing file carving. The developed OpenCL GPGPU processing framework was found to be more efficient than CPU counterparts when searching for greater amounts of patterns within data. This research also refutes claims that file carving is solely limited by the performance of the storage device, and presents compelling evidence that performance is bound by the combination of the performance of the storage device and processing technique employed.

TABLE OF CONTENTS

List of Figures

List of Tables

Definitions

Digital forensics: A field of forensic science that encompasses the recovery and analysis of data found on digital devices.

Forensic image: An exact digital copy of the data held on a digital storage device.

Digital evidence: A corpora of electronic data that contains information that may be of forensic interest to a digital forensic investigation.

File carving: The process of reassembling digital files from large unstructured streams of electronic data.

String searching: The act of searching for a combination of characters, or words, within a larger body of text.

Pattern matching algorithm: An algorithm that systematically performs string searching through a sequence of operations and rules.

File system: A record used by computers to store and retrieve data held on storage devices.

File fragmentation: The term given to a file that may be stored in more than one physical area of a storage device.

Central Processing Unit *(CPU)*: The main processing component within a computer that executes instructions required by a computer program.

Graphics Processing Unit *(GPU)*: A specialised electronic component within most modern computers that is designed to process large amounts of information quickly in parallel.

Integrated Graphics Processor *(IGP)*: An often scaled down version of a graphics processing unit that coexists as an integrated component on most recent central processing units.

Algorithmic Processing Unit *(aka: Processing core)*: A vital component of all processors that allows the processor to processing data with a set of instructions.

Cache: A hardware or software component that stores data so that future requests for that data can be served faster.

Computer Bus: The communication channels used to transmit data between components of a computer.

Speedup: The improvement in speed executing a task on two similar architectures with different resources.

Chapter 1: INTRODUCTION

1.1 Motivation

The goal of a digital forensics (DF) investigation, like any other investigation, is to uncover and present the truth (Casey 2011). DF can be defined as the science of ascertaining, analysing, and presenting digital evidence recovered from an electronic device, while DF investigation, on the other hand, aims to reconstruct a sequence of events that may have transpired from the digital evidence recovered. In recent criminal cases, DF investigations are of paramount importance when investigating any crime where electronic devices may have been used. DF investigations start from the moment an electronic device is discovered at the scene of a crime. Authorities have specific procedures and guidelines to ensure that any electronic devices and media are seized securely, safeguarding any data present on the devices from interference or modification. Any data held on electronic devices recovered is copied securely to a forensic image as part of the investigation process to ensure the actual device is unaltered from the state it was seized. However, due to the size of the data corpora being recovered in each investigation, it is becoming more conventional for forensic investigators to analyse the data directly from the drive with specialist write-blockers, a device which stops data on the drive from being altered.

Technology is evolving with each passing year, where today consumers have access to more types of devices – such as mobile phones, tablets, smart wearables and smart appliances – that are becoming just as sophisticated as a desktop computer, providing users smarter and easier access to entertainment, banking, communication and more. Unfortunately, alongside technology advancement, methods and devices which criminals are choosing to commit a crime have also diversified. With criminals more frequently using all manner of technology to facilitate crimes and avoid

apprehension, the amount of data gathered in criminal investigations is growing. The increasing volumes of evidence collected in modern digital forensic cases now challenge law enforcement agencies and researchers alike to advance techniques and technologies that are used to perform DF investigations and reconstruct electronic evidence.

The source of inspiration for this research originates from reviewing current literature in the field around what problems DF face in modern times. Garfinkel (2010, p. S64–S73) paints a bleak state of affairs that indicate that the successes that the DF community enjoyed for the last ten years are quickly coming to an end; that the tools and techniques used by DF professionals are being outpaced by the modern advancements in technology. This thesis is written in the context which primarily aims to address the problems faced by DF investigation carried out by policing authorities. Although, it is also with the hope that this research would also benefit all applications of DF recovery, including that used for personal and commercial use.

1.2 Problem

The growth of data storage available on modern storage devices has raised significant concern within the DF community, as current generation DF tools already encounter difficulty in processing modest-sized corpora of digital evidence within a reasonable timescale (Richard III and Roussev 2006, p. 76–91; Garfinkel 2010, p. S64–S73). This is mainly due to current techniques employed by DF tools to inspect each segment of forensic data held on storage devices seized in a DF investigation. Researchers have suggested moving processing intensive tasks to powerful purpose-built Beowulf clusters or super-computers (Ayers 2009, p. S34–S42) or distributing computationally intensive tasks amongst a group of machines (Roussev and Richard III 2004, p. 1–16). However, this research proposes to investigate the application of graphic processing units (GPUs) paired with parallel-friendly algorithms to compute processing intensive tasks— a

compelling alternative that may prove more efficient and cost-effective for DF professionals.

Modern GPUs can contain thousands of general purpose processing cores able to compute large amounts of data in a short time due to effective parallel processing design. Significant research has been conducted on utilising GPGPU programming on GPUs to provide aid in the calculation of highly demanding processing tasks. However, only some advances made in this research have been transferred into the field of DF, all which are primarily focused utilising CUDA— a closed-source programming framework presented by Nvidia exclusively for use on their line of discrete graphics cards (Marziale, Richard and Roussev 2007, p. 73–81; Skrbina and Stojanovski 2012; Collange et al. 2009, p. 1–10; Breß, Kiltz and Schäler 2013, p. 115–129; Chen and Wu 2013, p. 1–5; Zha and Sahni 2011, p. 141–158).

The introduction of central processing units (CPUs) with powerful integrated graphics processors (IGPs) in recent years from Intel (Intel n.d.) and AMD (Advanced Micro Devices n.d.) have opened up the possibility of powerful parallel processing without the requirement of discrete graphics hardware. Consequently, as these modern CPUs with IGPs are restricted from employing CUDA, utilising the cross-compatibility of an open-standards GPGPU processing framework – such as OpenCL – would be a logical step in tackling the processing demands of analysing extensive digital corpora on even modest specification computers.

1.3 Thesis aim

As modern storage technologies continue to improve the speed that data can be read from a storage device, the speed that DF tools can analyse data from storage devices have not. This research presents a study into methods to speed up DF analysis on modern storage devices. This aim of this research is to investigate whether the application of GPGPU technologies and modern parallelisable string searching

algorithms could speed up pattern matching—and by extension, reduce the time required to perform file carving on forensic data in DF investigations. This thesis aim can be broken down into the following research questions:

Q1: *"Could an OpenCL GPGPU framework provide a reliable foundation to analyse digital evidence and decrease the time required for processing forensic images without affecting accuracy?"*

Q2: *"Is there any benefit of employing multiple GPGPU processing devices to perform pattern matching on forensic data?"*

Q3: *"Are there any advantages of employing GPGPU processing over traditional CPU processing methods for performing pattern matching on forensic data?"*

Q4: *"Could further performance be gained through employing a multi-string search algorithm to perform string searching with the proposed processing techniques?"*

Q5: *"Is the potential processing rate in performing data analysis within the context of digital forensics limited by the speed of the storage device or the speed of the processor?"*

1.4 Thesis contribution

This thesis will present a comprehensive investigation into the possible benefits of employing modern GPUs and IGPs to conduct string searching on forensic data. The contribution of this research will build upon existing work around GPGPU processing

within DF, conducting experiments where OpenCL, an open-standards GPGPU programming framework, is used to analyse forensic data.

This research also investigates currently used algorithms used within DF, specifically analysing a well-established open-source file carving tool— Foremost (Kendall, Kornblum and Mikus n.d.). This investigation will determine whether currently employed algorithms used by Foremost are still suitable, or whether modern multi-string algorithms would be better suited for the requirements of modern DF investigation.

The resulting investigation of the above areas of string searching will then be used to measure whether the proposed methods could improve the overall performance when employed to conduct file carving. File carving is a technique used in DF where a forensic image is dissected for specific files that might be contained within its data. It can often provide a thorough and accurate view of all files contained within electronic evidence to DF investigators, including files which may have been deleted or stored in unallocated space of a computer's storage device.

The thesis aims to refute any claims that GPGPU processing has limited, or no, benefit to the problems faced by modern DF investigation. Results from this thesis aim to outline any performance benefits of applying modern parallel technologies over currently employed conventional CPU techniques. The advantages of introducing GPUs to handle the processing and analysis of forensic data are expected to succeed existing methods used in the field. The thesis makes an original contribution to DF research by being the first to investigate and analyse the benefits of applying OpenCL and the PFAC algorithm to the problems of DF investigation.

1.5 Thesis organisation

The remaining chapters of this thesis are organised as follows:

Chapter 2 explores the background required to grasp the topics presented in this research. This chapter will start by presenting a historical perspective on the current state of the field of DF. This will be followed by a technical walkthrough of; the concepts of string searching and file carving, an examination of the differences between traditional CPUs and modern GPUs, an introduction to OpenCL and GPGPU processing, and finally, presenting characteristics of pattern matching algorithms. The chapter will conclude with presenting related work in the field.

Chapter 3 will describe the methodology used in this research. This chapter presents the reasoning behind the chosen technologies and algorithms used to create the proposed solution. This chapter also outlines the adopted testing strategy, including how testing was conducted and what was measured. Finally, the hardware specifications of the test platforms are presented, which were used to conduct testing.

Chapter 4 presents the various case studies carried out during the length of this research. These case studies are similarly structured, outlining; the methodology behind each case study, the results that each case study produced and a discussion on what each case study showed. Finally, this chapter will summarise the findings of each presented case studies and analyse the significance of the results submitted by each.

Chapter 5 concludes by answering the research questions and presents future work intended on the expansion of this study.

Chapter 2: BACKGROUND

2.1 A historic perspective on the current state of digital forensics

The recognition of DF as a profession and scientific discipline can be traced back to the late 1980s and early 1990s when policing authorities set up specialist groups focused on investigating the technical aspects of computer related crimes (Casey 2011). Similarly, in the same era, multiple countries started introducing new laws that outlined clear guidelines for computer-related crimes where existing laws failed to prosecute against, these laws included dealing with issues such as; copyright, privacy and harassment, and child pornography (Crown 2008, p. 16; Nugent 1995, p. 159–182).

Since the field's early conception, DF matured through the 1990s with research and development of new tools and methods to facilitate the scientific acquisition of digital evidence; however, proper standards outlining the best practices to train and perform digital evidence seizure and investigation were not developed until the 2000s. Most notably, in 2002, the Scientific Working Group for Digital Evidence (SWGDE) published guidelines outlining best practices in the field (Scientific Working Group on Digital Evidence 2002). Comparably, in 2005, there were efforts to develop the examination of digital evidence into an accredited discipline under international standards (International Organization for Standardization n.d.).

While these international standards have been further interpreted by most countries and shaped into localised practical models outlining procedures to conduct DF investigations, such as the National Police Chiefs' Council (NPCC) guidelines adopted by the policing authorities within the United Kingdom, which replaces the Association of Chief Police Officer (ACPO) guidelines (Chief Police Officers 2008, p. 72). All guidelines on how to perform DF investigations arguably share the same core principles— that is that all information must be authentic, reliably obtained, and admissible. These

requirements also aid enforcements of strict requirements and standards which DF software tools used by DF investigators must abide by, ensuring that any evidence produced by these tools can be reconstructed using the same means to be admissible as evidence in a DF investigation.

Since its inception, DF tools have been known to keep ahead of the technological curve as they were capable of analysing modest sized corpora of forensic data associated with DF cases of that time; however, this opinion has changed in recent years, as concerns have been raised within the community around the lack of innovation and evolution of DF tools to cope with the increasing demand and volume of cases involving digital evidence. Paired with the lack of an effective research direction, some researchers are arguing that the "golden age [of digital forensics] is quickly coming to an end" (Garfinkel 2010, p. S64–S73).

A report was published on the conclusions of the Colloquium for Information Systems Security Education (CISSE) summit in 2009. The summit gathered a group of DF researchers, educators, and practitioners to discuss ideas for the developments of research and education within the field of DF (Nance, Hay and Bishop 2009, p. 1–6). It was identified that the current developments in the field have been largely a credit to practitioners of the trade. As a result, the tools that were developed have been in reaction to a particular niche set of scenarios or issues faced. This response-driven development cycle was seen by the panel as a danger, with the risk that DF methodology and associated tools eventually would lag behind the advancements of modern technology without adequate research efforts focused on advancing key areas. It was clear from the summit that this could have been drawn down to the absence of any research or development plan, and a lack of guidance for academic students to focus on in this ever evolving field.

The systematic review from Raghavan (2013, p. 91–114) also highlighted the need of DF triage tools to allow investigators to quickly analyse data corpora and present a high-level overview of the contents of forensic data. It is suggested that the ability to

conduct triage on large amounts of data will provide investigators with the ability to prioritise analysis of data that could be of key importance to a case. A thesis by Hales (2016) supports this claim, concluding that – in some scenarios – visualisation tools can help the investigator draw more accurate conclusions than what is achievable with traditional textual based tools. The ability to perform effective triage relies heavily on two predominant research areas, the ability to analyse large amounts of data quickly, and the ability to visualise data in an easy-to-decipher format for the DF investigator.

"In the decade since the inception of first generation tools, the limitations of this architecture have become apparent," quotes Ayers (2009), who identifies the constraints of the current generation of tools at processing large amounts of digital corpora. The author offers criticism on the existing trend of incremental updates to existing first generation of investigative tools, such as EnCase and FTK. Calling the tools "Generation 1.5" due to their lack of addressing significant limitations, and further failing to employ the necessary ingredients that the DF community desperately requires to stay ahead of technological advancements.

The problem of coping with the increasing volumes of digital evidence is not the only challenge that is faced in modern DF investigation. Advances in full-disk encryption have posed an insurmountable challenge to conduct any post-incident DF analysis unless the key used to encrypt the drive is known. As full-disk encryption is becoming commonplace in modern consumer operating systems – including Microsoft Windows, Apple OSX, and many popular distributions of Linux – there has recently been a gradual shift of research focus to investigate what evidence can be collected during, or prior to, a crime being committed. Naturally, this has called for improving the capability of network forensic techniques.

An example of network forensics is deep-packet inspection (DPI). DPI was originally conceptualised to allow internet service providers (ISPs) to analyse and optimise the flow of data transferred on their network. However, modern applications of DPI allow for data mining—revealing exactly what data is being requested over

networks (Dharmapurikar et al. 2003, p. 52–61). DPI conducts data mining through pattern matching, comparing live data sent over the network against a catalogue of known patterns of illegal or unwanted data. In practice, ISPs typically utilise data mining approaches with DPI to enforce policies on illegal material, however, state governments have been accused of using DPI for surveillance and internet censorship (Wagner 2009).

Intrusion detection systems (IDS) are another network devices employed to perform network forensics. IDS systems are typically employed on a local network to monitor live network traffic for the existence of any anomalies in the data transmitted (Vasiliadis et al. 2008, p. 116–134). Whilst primarily used to detect the presence of malware or unauthorised access on a network, the role of IDS systems could be extended to analyse live data for events of forensic significance on the local network—such as unusual usage patterns, or increased file sharing activities (Sommer 1999, p. 2477–2487).

Undeniably, utilising network forensic tools and expanding their role for proactive DF investigation could reap evidence that could be useful to an investigation, however, both approaches are surrounded by privacy, legal and ethical challenges that limit the amount of useful information that could be gathered by network forensic techniques (Khan et al. 2016, p. 214–235). Asides from these nontechnical issues, network forensics also share a mutual technical problem with traditional DF tools, as monitoring network traffic is typically a processor-bound activity that requires efficient processing frameworks to monitor live real-time traffic with minimal latency. As such, it could be argued that the greatest challenge faced by modern DF is the lack of efficient processing frameworks that can process the typical data associated with computing of the modern era.

2.2 An introduction to the concepts of file carving

File carving is the process of extracting a collection of data from a larger data set. File carving techniques are often used to discover and reconstruct files from data contained by storage devices, often when a file system's directory is missing or corrupted. In several DF cases, it was found that the recovery of deleted data or partial file data could greatly aid an investigation, which gave rise to the necessity of file carving (Raghavan 2013, p. 91–114). Recognised as "a science and an art unto itself" (Altheide and Carvey 2011), file carving reconstructs files by attempting to recognise content or structure of file types from an otherwise unstructured stream of data. The technique can be used to search for files on any file system type or device, as the file system is not used during the process; instead, data is interpreted in a raw form and searched sequentially for residual data that match the characteristics of certain files. Providing that the data held on the storage device not be encrypted, overwritten, or securely deleted, files can be reliably reconstructed through file carving techniques (Merola 2008, p. 40).

File carving is most effectively used in criminal investigations where files would often be obscured through some means, such as within hidden partitions, or through deletion. Through employing file carvers, DF investigators are often able to recover greater amounts of evidence in cases than relying on logically searching the files contained within a storage device's file system.

In its most basic form, file carving uses file headers and footers to identify files from the stream of forensic data. File headers and footers used are certain patterns of bytes which simply mark a location where a file begins and ends on the storage device. This simplistic method of file carving can often reconstruct a copy of discovered files from assembling the data between each header and footer found. However, it assumes that the files searched for are not fragmented, that the beginning of the file is intact and present, and that the file headers searched for are not a common pattern of bytes (Beek 2011). Files recovered that don't possess all of these assumptions may be unusable or

incomplete. Unusable and incomplete files are often referred to as false positives in file carving results as they often cannot be interpreted by the investigator.

Aside from the role that file carving plays in DF, file carving also has vital roles in other computing fields, such as personal and commercial data recovery on damaged hard drives. Several data recovery programs (WiseCleaner n.d.; File Recovery Ltd. n.d.; Grenier n.d.; Piriform n.d.; EaseUS n.d.) include, and rely on, file carving methods to restore files when a storage device's file system is damaged beyond recovery. These commercially available programs allow users to recover files from a storage device which would otherwise be lost.

File fragmentation is known to be one of the largest problems faced when performing file carving, as tools fail to detect whether a file had been saved into more than one location of the storage device. In recent years, file fragmentation has been one of the prominent areas of DF research. The DFRWS File Carving Challenge (Carrier, Casey and Venema 2006) challenged researchers to produce algorithms to detect fragmented files with minimal false positive rates. In response, research from Garfinkel (2007, p. 2–12) evidenced that files are rarely split into more than two fragmented pieces after conducting extensive fragmentation research on more than 300 used hard drives. Interestingly, Garfinkel also noted that most files of forensic interest are not typically fragmented. Findings from this research raise a compelling argument on whether developing smarter algorithms would benefit DF investigation when it may introduce the risk of ignoring some valuable evidence.

File carving performance relies heavily on string searching algorithms to accelerate searching through forensic data for patterns— as searching data is arguably the most computationally complex task involved when performing file carving. Although we traditionally associate string searching as a method of searching for particular strings in bodies of text, the concepts of string searching can also be applied to other areas of search, such as the current problem of searching for bytes within data.

2.3 Introduction to string searching algorithms

String searching could be considered to be one of the most important subjects in the wider domain of text processing. String searching algorithms are one of the fundamental components employed in lots of software and operating systems as a technique to perform the searching of one – or more – patterns within a body of text. String searching is used in a wide range of scientific fields where the processing and analysis of large volumes of data is required. As the typical amounts of data handled by many computational sciences arguably tend to double in size every eighteen months, research around string searching algorithms continues to provide theoretical computer scientists challenging problems to overcome (Charras and Lecroq 2004).

There are two forms of string searching algorithms; approximate and exact. The form of string searching algorithms that this research is interested in are exact string searching algorithms. This type of algorithm deals with absolute, rather than approximate, matching of patterns.

The effectiveness of string searching algorithms is typically measured by using computational complexity theory. Computational complexity theory is the study measuring the scalability of algorithms, and allows representation of how the time required to solve a problem grows as the input grows. The concept of growth is represented through the use of big O (O) notion. Through utilising this notion, the scalability of an algorithm can be presented without the added considerations of processor speed, programming language, machine architecture, and other factors (Mohr n.d.). Use of computational complexity theory will be used in this research to present the effectiveness of the algorithms discussed.

There have been copious amounts of string matching algorithms over the years, most which were developed in response to a particular problem. Alongside it, there have been many attempts to categorise algorithms based upon their searching

characteristics. This section will introduce three categories of string matching algorithms which will sufficiently cover the topics presented later in this research.

Brute force algorithms, also known as naïve search algorithms, presents a rather straightforward approach in performing string searching for patterns within text. Brute force algorithms typically operate by analysing each position within the text individually, attempting to match the following sequential characters with the searched pattern for a match. Arguably, brute force algorithms are the easiest to understand as it follows a very humanistic approach to finding patterns in text; however, this form of algorithm typically is found to perform slower than others as the time to analyse every position within the text is arguably ineffective for many applications.

Single string searching algorithms are a general classification of algorithms which incorporate optimisations to accelerate searching for a single pattern. These optimisations typically reduce the amount of text being analysed through employing a variety of pre-processing techniques; for instance, the Rabin-Karp algorithm (Karp and Rabin 1987, p. 249–260) utilises hashing to accelerate searching by hashing the searched pattern and parts of the text and comparing the derived results. The Boyer-Moore (BM) algorithm (Boyer and Moore 1977, p. 762–772), on the contrary, utilises what is known about the pattern to skip positions within the text where the pattern cannot be matched— accelerating searching performing and lessening the computational work required to analyse text.

Multi-string searching algorithms are defined by this research as an algorithm which can find a finite set of patterns. Although most brute force and single string searching algorithms could be modified to handle searching for multiple patterns, the effectiveness of these algorithms could diminish. Finite-state automaton algorithms, such as the Aho-Corasick (AC) algorithm (Aho and Corasick 1975, p. 333–340), employ the use of state machine logic to instruct the computer on how to proceed with each character read from text. Unlike other forms of string searching, finite-state automaton

string searching algorithms can handle the searching of multiple patterns with little, or no, performance degradation.

2.4 Differences between CPU and GPU architecture

This section will introduce both CPUs and GPUs and how the underlying architecture of both of these processors vary. While both CPUs and GPUs are very complex in architecture – consisting of a wide array of components which accelerate the processing of data – this introduction will focus on the essential features relevant to this study— arithmetic logic units (ALUs), cache, and computer bus. ALU cores – often referred to as processing cores – are where instructions are processed. These cores vary in complexity and can include various additional functions to aid the processing of complex tasks, such as performing encryption and decryption on data. The cache is typically a small area of memory embedded on the processor for storing data actively being processed and the resulting processed data. Lastly, the computer bus – sometimes referred to as a bus – is the communication channel that the processor has with the main system memory.

Throughout technological evolution, the design and implementation of CPUs have changed drastically, growing more complex in design and functionality; however, the fundamental purpose of CPUs have remained largely the same. Modern mainstream consumer CPUs typically range from two physical ALU cores (Intel n.d.) to eight physical ALU cores from high-end offerings (Intel n.d.). Some CPUs produced by Intel also employ Hyper-Threading; a proprietary technique belonging to Intel which allows each physical ALU core to host two virtualised cores that allow multiple tasks to be performed at once on one physical core— which Intel claim that enables the computer to make better use of the available resources on the CPU (Intel n.d.). An IGP is a frequently seen integrated feature of modern CPUs. IGPs are commonly a substantially large component on modern CPU chips, which as can be seen in figure 1, and consist of similar features that can be found on a discrete GPU.

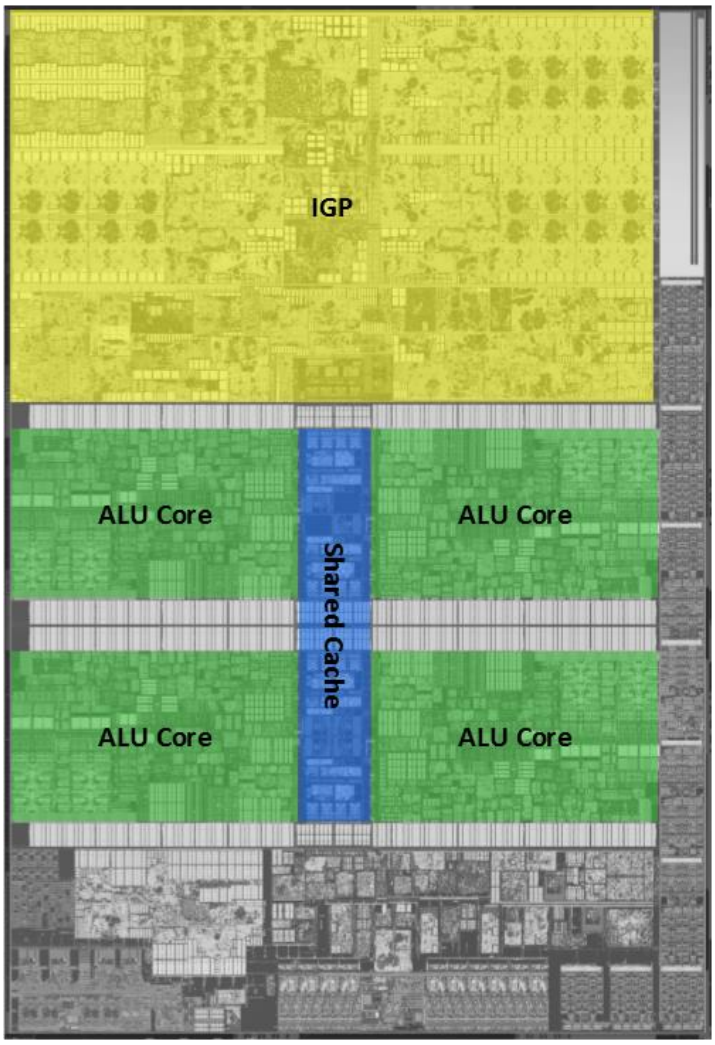


Figure 1: Intel Skylake CPU architecture

The ALU cores possessed by the CPU typically host an extensive array of arithmetic, bitwise and bit shift instruction operations, which can process a variety of tasks with ease. It is common for CPUs also to possess a significant amount of specialised operation instructions that can handle complex tasks with minimal effort. The additional specialised operation instructions found on CPU ALU cores ensure that common processing tasks requested by software and operating systems are done efficiently with less computational effort from the processor. ALU cores on the CPU are typically found

to be paired with a large shared cache for temporarily storing data from the main system memory. The cache of the CPU is usually large to cater for processing of more multifaceted data.

Concluding, CPU architecture is optimised for perpendicular processing of complex tasks with minimal latency; however, the architecture is not well equipped to handle large sets of superficial calculations on data due to its small quantity of ALU cores.

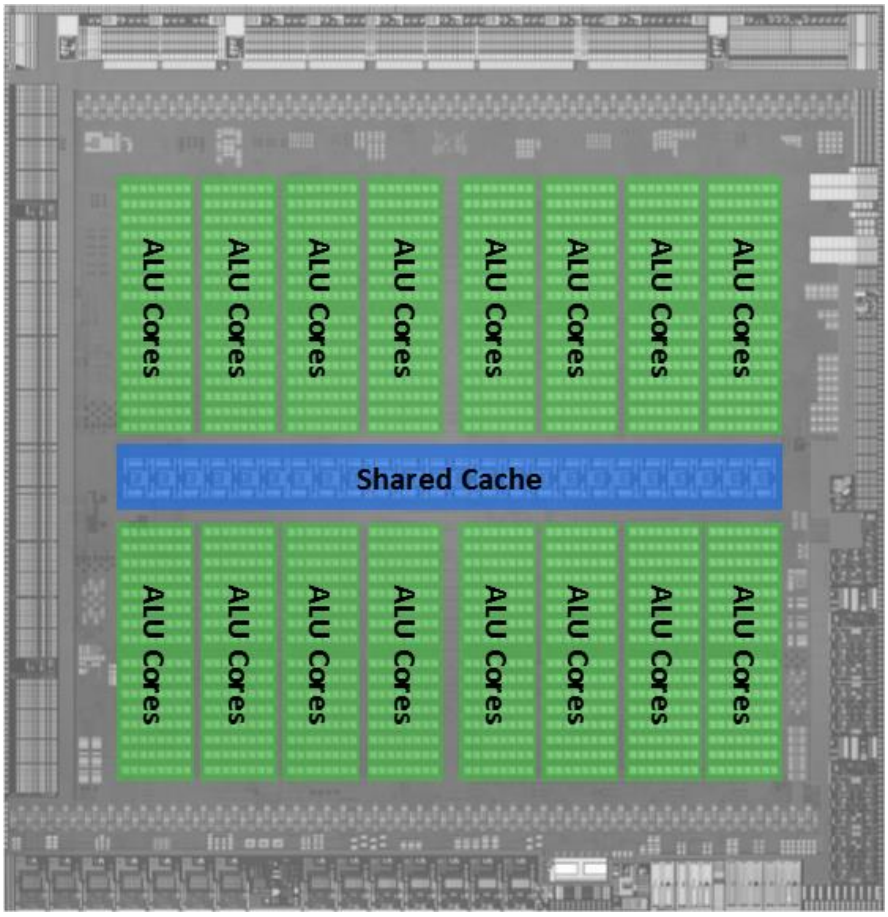


Figure 2: Nvidia Maxwell GPU architecture

In comparison, discrete GPUs – such as the Nvidia GeForce GTX 980 (Nvidia n.d.) depicted in figure 2 – possess thousands of general purpose ALU cores. Groups of ALU

cores form stream processors that can perform high magnitudes of highly intensive parallel calculations. ALU cores found on GPUs are far simpler in design to that typically employed of CPUs, and possess a limited algorithmic instruction set designed for the mathematical demands of graphical processing tasks; yet, due to the sheer volume of ALU cores that discrete GPUs employ, they are significantly faster than CPUs when used to compute simplistic arithmetic tasks.

The underlying processing model is the defining characteristic difference between CPUs and GPUs. GPUs employ a single instruction multiple data (SIMD) processing model, where its many ALU cores are used to perform the same operation on multiple data points simultaneously. While the SIMD processing model excels at sequential processing – performing simultaneous parallel computation with a single instruction – the model cannot process data concurrently with multiple instructions.

The ALU cores of the GPU are typically found paired with a smaller cache than CPU counterparts that hinders the ability to handle complex datasets. The small cache is a characteristic by design as GPUs do not benefit from having a large cache for typical graphics workloads. The disadvantage of having a relatively small cache on discrete GPUs for compute purposes, however, is offset by the fast data transfer rate of the bus from the system memory to the processor's dedicated memory.

Discrete GPUs possess a vast store of dedicated memory to hold data. The memory found on discrete GPUs is characteristically optimised for transferring high volumes of data at low latency between memory and the discrete GPU's processing units. However, to utilise discrete GPU memory, it requires the transferral of data from the main system memory to discrete GPU memory. This data transfer required by discrete GPUs is an additional step that is not needed by CPUs and IGPs, both of which directly utilise the main system memory to read and store data. Nonetheless, the transferal of data from main memory to discrete GPU memory is not a particularly timely operation to do due to the high bandwidth bus typically found to exist between the two memory locations.

The architecture employed by discrete GPUs – by design – is irrefutably better suited for handling large quantities of simple but intensive calculative loads. Respectively making discrete GPUs not only exceptional at fulfilling its primary role of processing graphical and physics based instructions, but also arguably superior at assisting in the scientific calculation of processor intensive numeric datasets than CPU.

2.5 Understanding OpenCL and GPGPU processing

OpenCL is a heterogeneous open-standards GPGPU programming framework that is managed by Khronos Group (n.d.), a non-profit technology consortium. The GPGPU framework is widely compatible across a variety of devices offered by various vendors, including GPGPU devices from leading hardware vendors such as Intel, AMD, and Nvidia. OpenCL allows applications to perform multiple levels of parallelised processing across one, or more, processing devices— allowing programs that employ OpenCL the ability to utilise the full range of processing power available on the computer.

GPGPU processing adopts elements above traditional programming languages which normally operate through executing each command in a perpendicular or limited threaded parallel fashion. As GPGPU processing languages employ the use of a system's GPGPU device – such as the GPU or IGP – to perform processing on a massively parallel basis, additional code in the form of a kernel is required. In principle, a kernel is a set of instructions which direct the GPU on how to process data. The instructions that form GPGPU kernels tend to be far more restricted in functionality— only offering logical and arithmetic functions.

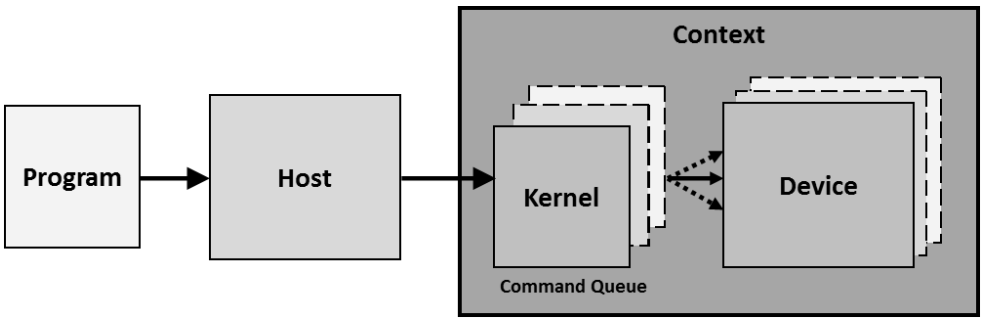


Figure 3: OpenCL processing methodology

A brief overview of the processing methodology with OpenCL is outlined in figure 3. In this analogy, the program serves as the main program binary written in a programming language; such as C#, Java, Python, etc. The host refers to the computer that processes the instructions provided by the main program. The context refers to OpenCL specific instructions which are provided by the host from the program. Inside the context, kernels dictate the instructions sent by the host through the program that instructs GPGPU processors on how to process data. Devices within the context refer to the OpenCL compatible processors that the kernels are sent to.

In a program, there may be multiple kernels required to execute a series of tasks on GPGPU devices, a group of kernels are defined as a command queue. Likewise, a context may have many GPGPU devices available to process the command queue of the program. Each GPGPU device used can only have one command queue.

To help explain the process and how each of the five elements work, a walkthrough of a simple addition calculation program ($c = a + b$) execution will be described. When executed, the program initially sends a list of instructions for the host to carry out. Within the program instructions, it outlines two integer arrays of numbers that require to be processed (arrays a and b). Firstly, the program assigns a context to process the arrays of numbers – in this example – a singular GPGPU device. The program then instructs the host to copy both integer arrays a and b from the host's main memory to the GPGPU device memory. Following this, the program also allocates space on the

GPGPU device memory to store the results (array c). After all the relevant data is loaded onto the GPGPU device, the program then instructs the host to load and run a kernel on the device. The kernel loaded onto the GPGPU device simply instructs the processor to add each entry of arrays a and b and store the result in array c within GPGPU device memory. When the GPGPU device has finished executing the kernel, the host copies the results contained within array c from GPGPU device memory to the host's main memory. Once retrieved, the program has finished processing and should have an array of calculated integers.

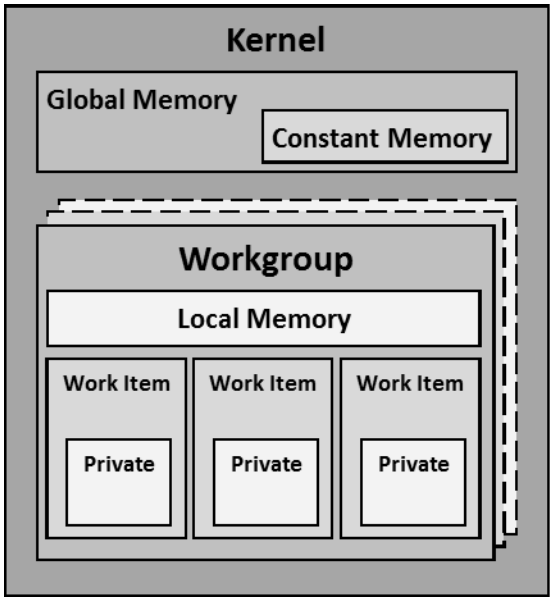


Figure 4: OpenCL kernel memory model

To achieve a full basic understanding of GPGPU processing, this research will outline the key memory components specified within an OpenCL kernel. Figure 4 illustrates the OpenCL kernel memory model that can be translated to compatible parallel processing devices. Within this diagram, there are a few memory areas of importance in the kernel for processing large quantities of data. The biggest memory location available on the GPGPU device is the global and constant memory – these

memory locations can be read by all the GPGPU device's computation units. The global and constant memory location shares similarities with the main memory of a computer and the relationship it has with the CPU.

Global memory signifies the memory which can be directly accessed by the host, and can be used to send and receive data from GPGPU devices. Constant memory shares the same function as global memory, with the exception that it is only used to store static variables. In the previous simple addition program example, the integer arrays would have been stored in global memory, so that all processing units could access the required data to compute and store. Additionally, once the integer arrays were processed, it allowed the host to read the GPGPU device's global memory to retrieve the results.

Within each workgroup resides two forms of memory. The first is called local memory. The local memory on each processing unit tends to be notably small in size. The primary purpose of the local memory is to serve as a scratchpad for the processing unit, such as temporary storing intermediate calculations related to processing. All work items can make use of this memory to store variables.

Storing a counter could be an obvious example of where utilising local memory is vital; as if different workgroups are trying to change the same counter held in global memory, the counter could be accessed and changed at the same time by several processing units— potentially causing inconsistent and unreliable results. However, if each workgroup counted within their local memory, the values would only be accessed and changed by their belonging stream processor. This allows an accurate count record, which can then be combined with other stream processor counts once processing ends to provide an accumulated value on completion.

Private memory is the second form of workgroup memory that serves as unique storage dedicated for each work item. Typically, this memory cannot be accessed by default and serves as the storage space for processing each work item.

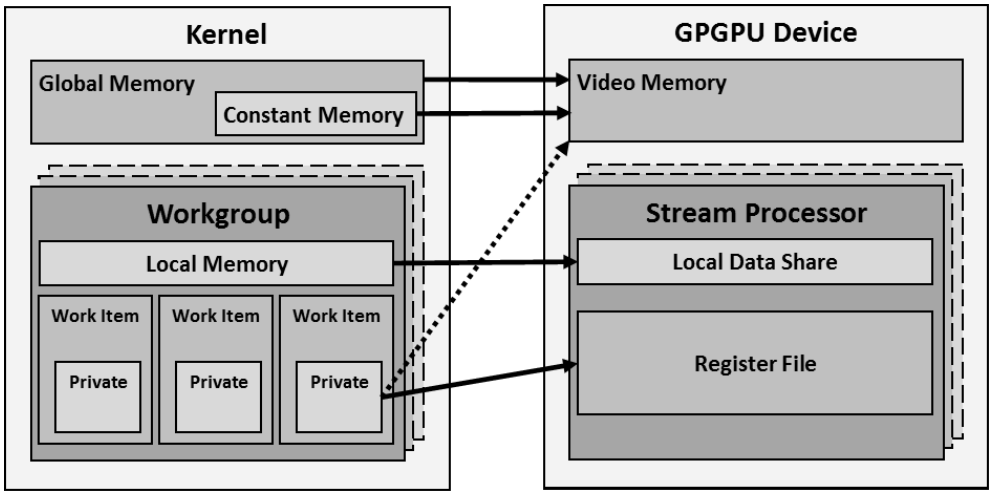


Figure 5: OpenCL kernel to GPGPU translation

Figure 5 shows how the OpenCL kernel memory model translates seamlessly to GPGPU hardware. Global memory and constant memory reside on the GPGPU's primary memory, which could be the dedicated memory on a discrete GPU, or if using an IGP, a dedicated area of main memory. The local memory records to the stream processor's local data share. Lastly, private memory maps to the stream processor's register file. However, if the work item requires to work with a private array or an oversized register, it should be assigned to the GPGPU's primary memory instead due to memory capacity constraints.

This concludes the basic concepts of GPGPU processing and the underlying processes. Additional recommended reading on OpenCL and processor architectures can be found within the pages of Heterogeneous Computing with OpenCL (Gaster et al., 2012).

2.6 Related work

Whilst there have been many studies of the benefits of utilising GPU and parallel processing in research areas where the processing of significant amounts of data is

paramount (Pungila and Negru 2012, p. 354–369; Wu 2013; Bellekens et al. 2014, p. 295–301; Vasiliadis et al. 2008, p. 116–134; Bellekens et al. 2013, p. 5; Haseeb 2013, p. 9–14; Bhamare and Banait 2014, p. 24–28; Kouzinopoulos et al. 2015), there has been little research that investigates what benefits DF could reap by employing such methods. The majority of existing research in DF is implemented using CUDA to accelerate DF searching. While the research incorporating CUDA shows clear performance advantages (Karimi, Dickson and Hamze 2010, p. 12; Fang, Varbanescu and Sips 2011, p. 216–225), CUDA suffers greatly from being incompatible with GPUs from vendors other than Nvidia. This may have been insignificant at the time of the research, but as it has become commonplace for modern computers to have access to both a discrete GPU as well as powerful IGPs onboard the CPU, there is little reason now to choose CUDA over an open-standard alternative like OpenCL which would make full use of the computer's full computational power.

The papers reviewed in this section are the most complete in investigating the application of GPU processing within the field of DF. The studies highlighted here argue both for and against the application of GPUs to tackle the processing issues of modern DF investigation; however, most papers suffer a common inadequacy in failing to provide enough information to reproduce experiments for validation. More importantly, the lack of detail from the papers renders it difficult to transition their research to create beneficial tools that could be used by the DF community.

A theoretical insight was presented by Skrbina and Stojanovski (2012) which discussed the preparation and processes involved in creating a GPGPU solution to accelerate file carving. The authors explored how CUDA could be utilised within the context of DF investigations, examining how the different characteristics of string and pattern matching algorithms are suitable for GPGPU parallelisation. The authors conclude that the most appropriate algorithms for parallel applications are the BM (Boyer and Moore 1977, p. 762–772) and AC (Aho and Corasick 1975, p. 333–340) algorithms for handling single- and multi-string searches, respectively. However, the

study failed to mention modern algorithms, including recent adaptations of the AC algorithm, which were specifically designed for parallel execution.

Richard and Roussev (2005, p. 1–10) were amongst the first to apply parallelism to DF investigation by presenting a paper outlining the requirements needed to reproduce high-performance file carving. As part of the author's research, they present an open-source tool called Scalpel, a parallel file carver based largely upon modifying Foremost— a well-established Linux file carver. Where Foremost only utilises a single core of the CPU to perform file-carving, Scalpel utilises parallel processing on all available cores to accelerate performing file carving on forensic data. The author's results prove that parallelism undeniably – yet unsurprisingly – yields much faster results.

A further empirical study conducted by Marziale, Richard and Roussev (2007, p. 73–81) expanded upon Richard and Roussev's earlier research by investigating how the use of CUDA GPGPU processing could accelerate Scalpel. The study compares the time taken to complete various searches through different sized forensic images using the unmodified and modified GPU versions of Scalpel. Results from the study show significant improvement with GPGPU acceleration, which clearly demonstrated that incorporating GPU technology is a practical option for significantly increasing processing performance in existing DF tools. At the time of the research, however, the CUDA framework was still in its early stages of development. The authors acknowledge that the beta release that was used for the study may have possessed some bugs, and further suspected that the compiler did not fully optimise the code; these factors may have limited the proposed solution's potential achievable performance compared to that which could be derived today.

Contrasting research from Zha and Sahni (Zha and Sahni 2011, p. 141–158) states that when incorporating a fast multi-pattern matching algorithm, the performance gain achievable from file carving is limited by the time required to read data from the disk ("disk-bound"), as opposed to the time needed to conduct string searching on the data.

The authors similarly conducted experiments through modifying Scalpel, where they incorporated a series of BM and AC algorithms to aid string matching on the CPU. The authors' experiments indicate that multi-threaded acceleration using a dual-core CPU did not improve the required processing time, concluding with an arguable assumption that there are no advantages of using other accelerators such as GPUs, despite presenting no actual experiments involving GPGPU processing. Later research from the same authors (Zha and Sahni 2011, p. 277–282, 2013, p. 1156–1169) shows that incorporating similar algorithmic techniques using GPGPU processing produced notable improvements over single-threaded CPU approaches, surpassing multithreaded CPU processing in some scenarios. Results of these later studies from the authors form the argument that Zha and Sahni's earlier research on processing techniques to improve file carving had not been thoroughly explored.

Another interesting method of utilising GPGPU processing in file identification was adopted in a thesis by Mohan (2010), who utilised an MD6 file hashing method on a CUDA GPGPU framework to identify similar files individually and contained within an archive. His results demonstrate a significant performance increase over traditional CPU processing, which led to conclude that the parallel nature of GPUs is well suited for the large-scale processing of MD6 file hashing. Despite the author's findings, this method of discovery requires a list of known file hash signatures to search for, limiting its usefulness when performing an exploratory examination for unknown incriminating files.

Collange et al. (Collange et al. 2009, p. 1–10) demonstrated a similar but novel method of utilising GPU processing to aid file identification in DF investigation. The authors use a CUDA GPGPU implementation to calculate and compare hashes of data to identify potential image file identifiers located on storage devices. The authors concluded in their study that, with the computational power of the hardware, GPUs make an ideal platform on which to perform parallel hash calculations, potentially delivering a powerful and usable file identification technique for DF investigation.

Although Collange et al. eliminate the requirement of knowing complete file hashes by searching for file identifiers, the proposed approach still requires and is heavily dependent on the CPU to verify the matches found as valid image files; this potentially slows the overall performance when faced with forensic images containing significant amounts of data.

2.7 Chapter summary

This chapter has presented a background of the current state of digital forensics and the problem of DF tools failing to innovate. It was reported that current tools are failing to address significant limitations in processing large amounts of forensic data, suggesting the need for tools to incorporate more capable processing techniques. Also presented in this chapter is the basic knowledge required to comprehend the themes presented in the context of this research, including an introduction to the concepts of; file carving, string search algorithms, CPU and GPU architecture, and OpenCL GPGPU processing.

The chapter concludes by critically evaluating previous attempts of investigating the benefits of applying GPGPU programming to the field of DF. Existing research shows that there are significant areas of the field's current GPGPU research that would benefit from further investigation. It is hypothesised that further performance enhancement could be achievable through the careful application of GPGPU processing techniques and modern multi-string searching algorithms to the problem of file carving.

In the following chapter, this thesis will present how this research approached the problem. This section will include how a GPGPU solution was devised and the testing strategy used to evaluate the performance enhancements over current DF file carving tools at performing string searching and file carving.

Chapter 3: SOLUTION CONSTRUCTION AND TESTING METHODOLOGY

3.1 Research platform development

A software platform was created to test different methodologies in performing string searching and file carving of forensic data. While previous studies in this field used one of the freely available open-source file carving tools as a software platform to perform performance related research; it is anticipated that clearer data could be obtained from starting with a bespoke platform built specifically to measure performance. The developed software platform would ensure that both the CPU and GPU processes inherently follow the same chain of processing operations without the need to modify existing file carving software. Additionally, the software created could ensure that any processing undertaken was relevant to the task of performing string searching or file carving.

While the decision to build new software over modifying an existing file carving tool may seem to be far greater of an undertaking, there were clear advantages of building a new platform. Building a new platform provided complete control of measuring performance metrics and debugging, and allowed a modular design to be adapted— allowing the platform to be easily modified to trial different processing approaches as this research progressed. The software platform built to perform testing within this research will be referred to within the rest of this thesis as *OpenForensics*.

This section will outline some of the development choices made when creating the OpenForensics research platform. The source code of OpenForensics is readily available on GitHub at: *https://github.com/ethanbayne/OpenForensics*. The source code available on GitHub outlines the latest build of OpenForensics, which may have changed significantly since the time of writing this thesis.

As the research took a rather exploratory approach in each case study, OpenForensics was developed with an iterative software development cycle, producing several prototypes for analysis and performance comparisons at regular points during the research. Through prototyping, known and unknown performance factors were easier to identify, which overall aided – and developed – further research goals. The rest of this section will outline the development cycle that OpenForensics followed— design, implementation, and testing.

3.1.1 Technologies used

C# .NET (Microsoft n.d.) was used to build OpenForensics. The decision to choose C# was simply due to the language's ability to be easily used as a rapid application development platform with a wealth of analytical and performance metric tools. This enabled easy monitoring of currently running processes and allowed for painless debugging— factors that greatly aided development. It is recognised that C# admittedly falls short of the possible performance and cross-platform compatibility that could be achieved with other programming languages, such as C++. However, it is envisioned that the margin of performance gain measured by the final solution would not differ substantially from the choice of the programming language used. By building with the .NET framework, it was also easy to incorporate a simple GUI to help drive the research and allowed for more visual feedback during testing. This proved later to be invaluable during the testing of larger datasets utilising multi-threaded approaches as it provided effective feedback of what was happening on each processing thread as data was being processed. There were other advantages of choosing C# as a development environment, as it also benefits by adopting the .NET framework's memory management feature— garbage collection (GC). GC manages the allocation and release of memory used by the application; for string processing, amortised GC is more efficient than manual memory management.

To handle GPU instructions, CUDAfy.NET (Hybrid DSP n.d.) was utilised. CUDAfy.NET provides the necessary libraries that allowed the easy management of GPUs from within the C# software platform. Cudafy .NET provides a comprehensive set of libraries and methods to allow C# applications to interface with GPUs. The framework also hosts a wealth of emulation features, which simulate the processing of GPU kernels on the CPU, providing excellent simulation and debugging functionality. The most significant advantage of using Cudafy .NET within this research arises with kernel creation. Cudafy .NET comes with the ability to translate a kernel written in C# into OpenCL. Generated code from CUDAfy .NET for OpenCL kernels was manually validated in each case study for optimisation.

3.1.2 Research platform design

This section aims to provide a general overview of design choices taken with the OpenForensics framework, providing an insight into what features remained largely static across case studies. As each case study presented within this research varies in the method used to search through data, the various implementations will not be discussed here. Information about the processing framework implemented by each case study can be found in the methodology section of the respected case study.

Designing a new framework to carry out testing was done to isolate and control the identified processing tasks that GPGPU processing aimed to improve. Early in the research, it was decided that it would be appropriate for the research to develop and maintain a simplistic application that measured the performance metrics related to string searching, rather than refactor an existing file carving application. It was believed that this approach allowed the researcher to fully dissect other related processes that support string searching – such as file reading – and investigate methods of improving them.

As the research was focused on how GPGPU processing could aid DF investigation, identifying processing intensive operations was paramount to satisfy the design goals of OpenForensics. The initial design stages of OpenForensics revolved around analysing current open-source file carving tools – Foremost and Scalpel – and dissecting how they performed file carving. When investigating these open-source tools, there was an assumption that string-searching operations would be the most processor intensive task that file carving carried out. This assumption was later confirmed by conducting profiling against the open-source tools and analysing what processing loops within the code took the longest to process. It was found that string-searching was the most processor intensive task carried out.

Identifying processor intensive processes enabled the research to focus its development cycles around these aspects. Thus, string searching and file reading components followed a constant design, implementation and testing development cycle between case studies—much more so than any other component of OpenForensics. Case studies were often completed when significant improvements were obtained through newly implemented string searching and file reading features. The results of testing newly implemented features also fed design decisions for the next cycle of development.

During development, a front-end GUI was developed to aid the testing of the application. The front-end GUI went through only two iterations, from being a largely simplistic and unpolished interface that had largely hard-coded test options, to later being a more customisable interface that linked many of the variables for conducting string searching and file carving experiments. The latter changes to the interface being implemented for the purpose of external testing, providing investigators more control and customisability over the tests performed. The options for the interface were designed to incorporate the commonly used command line options of Foremost into the GUI interface.

The latest OpenForensics interface is illustrated in figure 6. The interface is split into a few sections. The topmost section containing inputs for any case and evidence references, which are used only to uniquely label each test. Below this, a hard drive or file selector to select what data should be analysed. This is followed by a search target selection, where file types or keywords to be searched for can be selected from a drop-down list. Lastly, there is a hardware platform selection, where it is possible to select what processor present on the system should be employed to conduct the string searching or file carving.

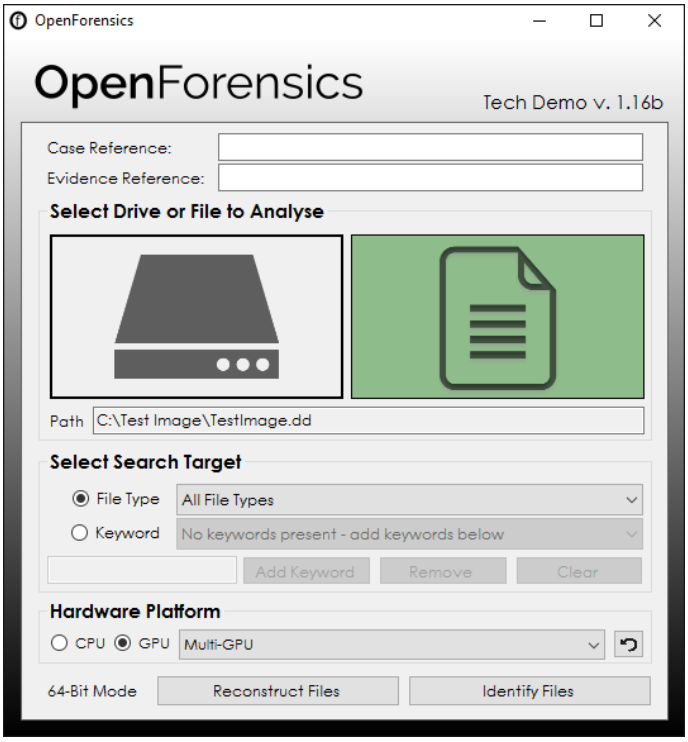


Figure 6: OpenForensics GUI interface

Developing a new framework allowed for processing monitoring tools easier to implement, such as the core activity monitor illustrated in figure 7. The core monitoring tool allowed for visual analysis of what each CPU, or GPU, core was doing at any given moment during testing. Having passive visual monitoring allowed identification of any

processing bottlenecks more apparent during testing. This was used in conjunction with profiling to aid the development of the processing frameworks.

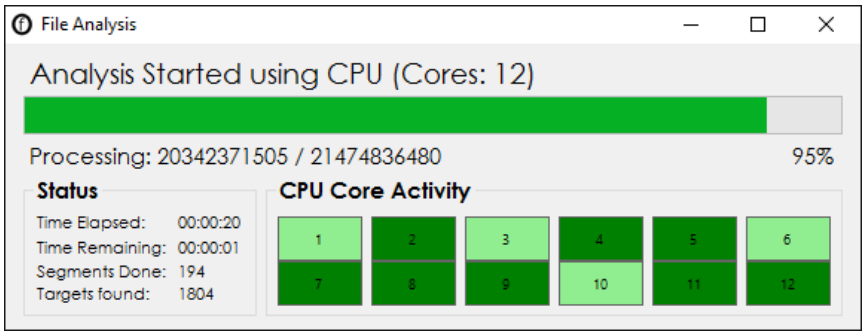


Figure 7: OpenForensics processing interface

3.1.3 Research platform implementation

In the implementation section, the author outlines the initial implementation of OpenForensics. This section was written as a supplement to the source-code available on OpenForensics repository. The research took an iterative approach to development and refactored various areas on response to the previous case study. As such, the implementation of OpenForensics focused on developing three classes; interface, analysis, and engine. The interface class contains methods relevant to the main GUI of the application, including test parameters and general program implementation functions. The analysis class comprises of procedures relevant to string searching or file carving operation and reading forensic data from disk. Lastly, the engine class has functions related to algorithms used to search, pre-processing, and interfacing with processors. Figure 8 presents an abstract class diagram to show the relationships that these classes have with each other. A more detailed class diagram of the final case study solution can be seen in appendix A.

The interface class, as the name suggests, is responsible for the OpenForensics front-end interface. The primary role of this class is to deal with the configuration of the

test parameters. Asides from the visual elements of the interface, the interface class deals with loading file-type settings from the XML configuration file. The class also populates information about the system and, if available, sets up multi-GPU processing parameters. The multi-GPU parameters identifies, and filters, all available discrete GPU and IGP devices on the system and allow these devices to be passed to the analysis class as a variable. The last responsibility of the interface class is to sanity check the selected options for the operation, ensuring that all inputs selected are valid before the analysis class is invoked.

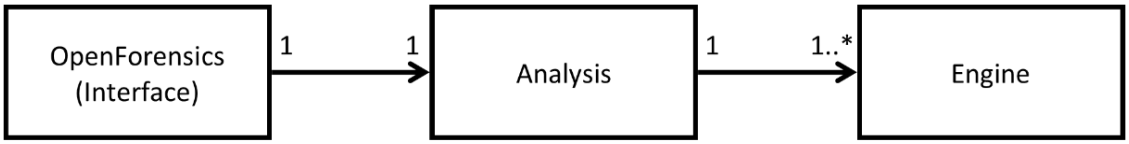


Figure 8: OpenForensics class diagram

The analysis class is responsible for a large proportion of the string searching and file carving tasks carried out by OpenForensics, including; file reading, processor thread initialisation, and allocation of data segments to the aforementioned processing threads for analysis. The analysis class also incorporated a very basic file carving operation. The ability to perform file carving was considered to be a low priority, as the primary goal of this research was to ascertain whether GPGPU processing would accelerate string searching in a DF context. As such, the file carving method used by OpenForensics – at the time of writing – is very basic when compared to the file-specific carving operations carried out by Foremost. OpenForensics adopts a rather naïve method of extracting files from data, by simply reconstructing the data found between a file header and a matching file footer.

The engine class instructed the processors how to process the data. This class contained three components. The first is the processor object initialisation that handles requests from the analysis class for a new processing threads to be set up. When

invoked, the processing thread initialisation will set up space in memory for the processing thread to load volatile data, such as the buffer for data read and counters for the results. The second primary role that the engine class had was the ability to do any pre-processing required for algorithms—such as the lookup table generation for the Boyer-Moore algorithm. The third, and most important, role that the engine class is responsible for is searching data. The engine searches data by utilising the algorithm declared by the analysis class.

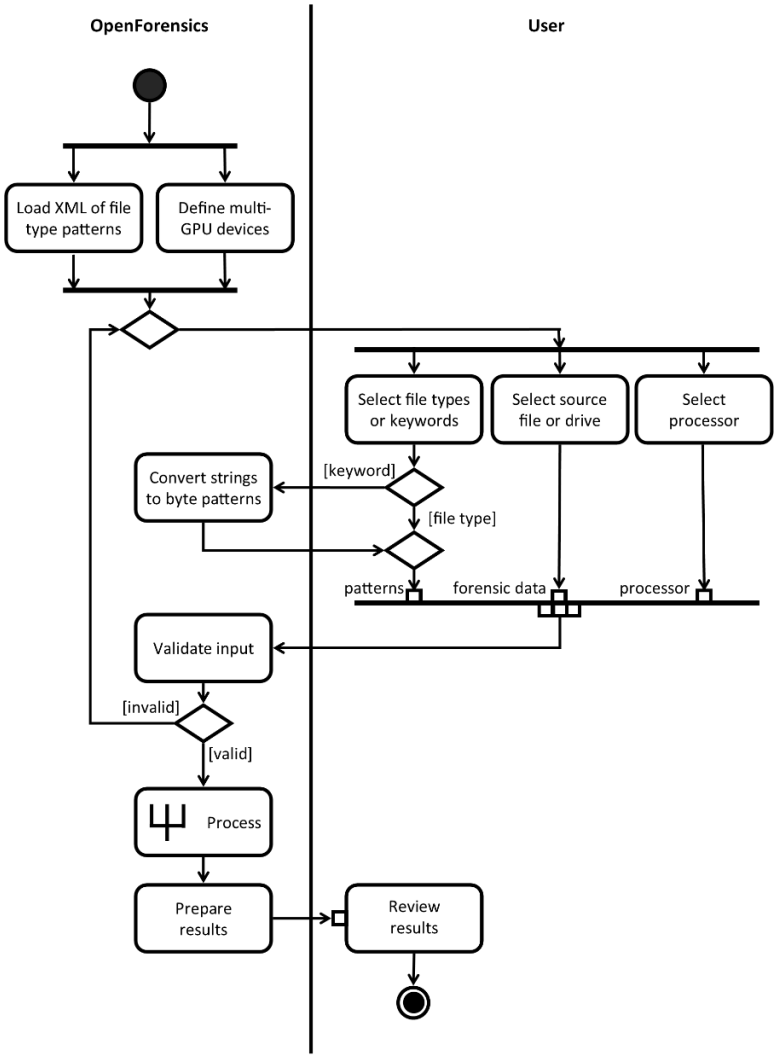


Figure 9: OpenForensics operation activity diagram

The XML configuration file loaded by OpenForensics on launch contains the necessary parameters needed for OpenForensics to conduct analysis with file types. The configuration file contains a list of file types, each with four properties. The first of the properties is the file type extension. The file type extension is used by OpenForensics for the file type interface menu and, if reconstructing found files, to set the file extension. The second property classifies what type of file it is—whether an image, video, audio document or miscellaneous file. The classification information is used to define groups of file types to allow easier batch analysis, e.g. to search for all image file types. The third property is the file header value, the byte sequence that marks the start of that file type. There may be multiple header values defined for each file type to cope with variations of file type headers. The fourth – and optional – property is the file footer value (*"EOF"*), a sequence of bytes that marks the end of that file type.

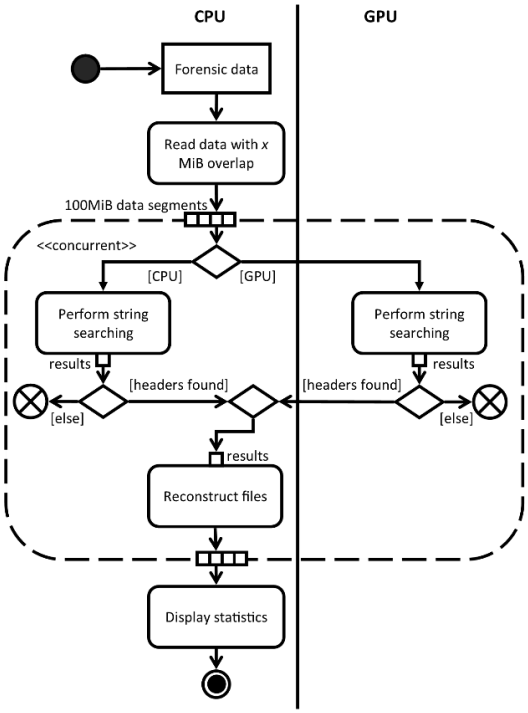


Figure 10: Case study 4 file carving process activity diagram

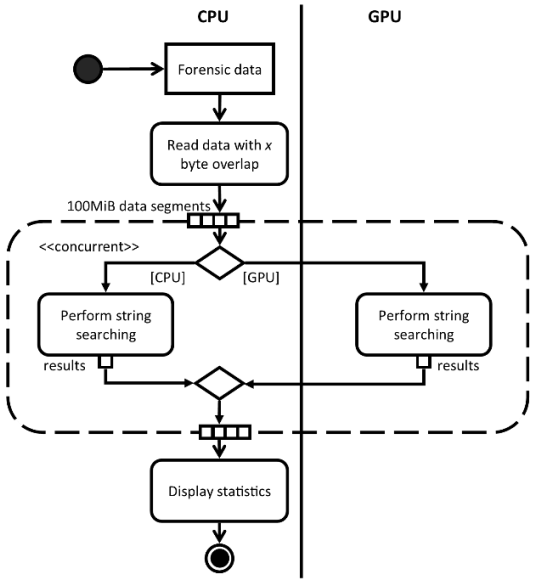


Figure 11: Case study 4 string searching process activity diagram

The operation of OpenForensics is as outlined in the activity diagram in figure 9. In this operation diagram, the results stipulate the output of the processing. Performing a string searching analysis will output what file types – or keywords – were detected in the forensic data. Whereas performing a file carve operation will report back the aforementioned and present files that were reconstructed from the forensic data—assuming that the file types specified have a valid header and footer in the XML configuration file. A processing activity diagram, based on case study 4, is outlined in figures 10 and 11—outlining an example of both string searching and file carving processes respectfully.

3.1.4 Research platform testing

The initial CPU implementation aimed to mimic Foremost in its approach by implementing the same modified Boyer-Moore algorithm. By doing so, early builds of OpenForensics focused on accuracy over functionality, ensuring that the developed string searching method would correctly identify file headers and footers. To ensure the compliance of this, the developed framework was tested against three forensic images—the 20GiB image used in this research, a 5.36GiB Windows XP image, and a 120GiB Windows 7 image. The latter two of the three images were artificially created by digital forensic educators to mimic realistic usage and used as forensic cases to teach DF investigation. Results from these tests with OpenForensics were compared with the results gathered by Foremost. It was assumed, and later confirmed, that results from the initial algorithm and Foremost would be the same as they utilised the same algorithm.

As development adopted an iterative development cycle, test cases were developed alongside OpenForensics, often in response to errors and inconsistencies found during development. Algorithm and pre-processing tests were validated through automation utilising a pre-set configuration file and comparison against expected results

from earlier trials, but also manually by the researcher by inspecting the forensic data with a hex editor. Testing was reinforced with frequent code reviews and refactoring exercises during development.

Before each case study, the accuracy tests above were rigorously followed to ensure the integrity of any changes made to the processing approach or algorithm used. On the occasions where there was a mismatch between the results obtained and Foremost, case studies were delayed until the problem could be identified, and rectified accordingly. As DF is a science, scientific standards were maintained as a paramount objective of the research. As such, case studies could only proceed when the parameters presented reported accurate findings.

3.2 Algorithm choices for data analysis

Careful consideration had to be done around what algorithms would be best suited to accelerate processing of forensic data. There have been plentiful amounts of research that aims to compare the efficiency of algorithms in processing large data within different fields (Gharaee 2014, p. 946–953; Lin et al. 2013, p. 1906–1916; Rasool and Khare 2013, p. 6–16; Arudchutha, Nishanthy and Ragel 2013, p. 231–236; Mokaram 2015; Soroushnia et al. 2014, p. 253–264), the problems and comparisons presented by this research are relatable to the problem faced within DF.

As GPGPU devices have powerful parallel capabilities, a brute force algorithm was adopted in early research to measure baseline performance of GPGPU solutions. The brute force algorithm – being the simplest of the algorithms presented within this research – operates by searching each byte of data sequentially looking for any potential pattern match. The time that the brute force algorithm will take to search for a pattern of length m within a data stream of length n is $O(n)$ in its best case where the first byte of searched patterns are not found, and $O(nm)$ in its worst case where each byte requires validation against the longest pattern.

As this research aimed to improve upon current DF tools, an investigation was done on how current open-source DF tools processed data. It was found that two popular Linux-based tools – Foremost and Scalpel – favoured the use of a modified BM algorithm for performing string searches. This research decided to replicate the BM processing method employed by these tools for CPU processing. This allowed the study to produce baseline performance metrics of the modified BM algorithm employed by the tools above, which could be used to compare against the performance gain of any proposed GPU implementations or alternative algorithms.

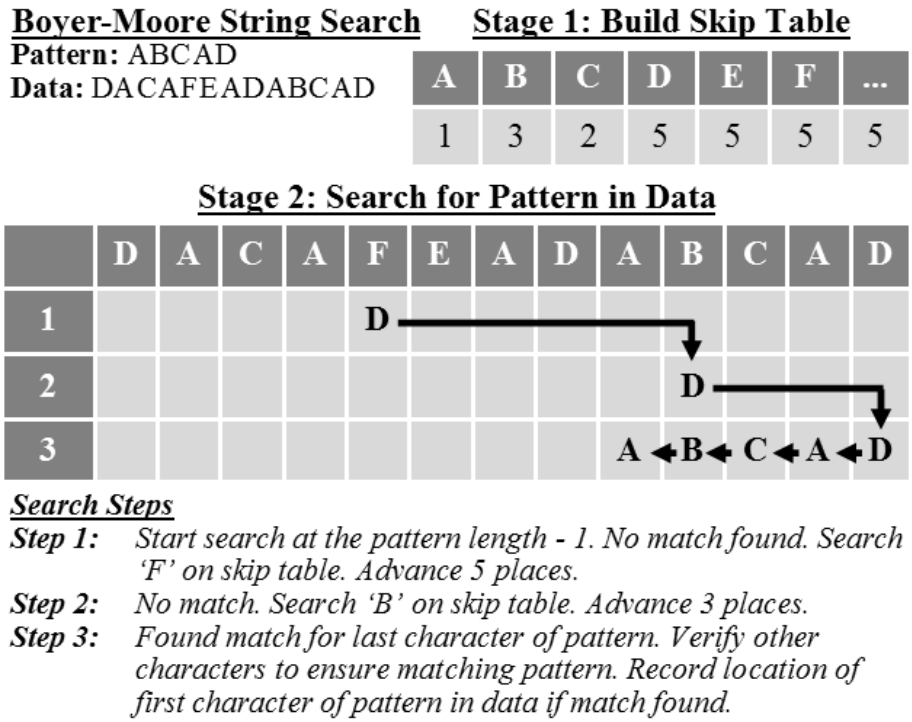


Figure 12: Boyer-Moore algorithm example

The BM algorithm operates by searching through a stream of data for the last byte of a pattern. When the algorithm discovers the last byte, the rest of the pattern is validated byte by byte. If the algorithm validates the complete pattern, the program will record an index on where that pattern was found within the stream of data before continuing to search through the rest of the data. Searching through the data stream is accelerated using a skip table. The skip table is created before the search begins, and acts as a reference on how far to move in the data stream depending on the value read, significantly reducing the search time required looking for a potential match. Theoretically, the time for the BM algorithm to find a pattern of length m inside data stream of length n is $O(n/m)$ time in its best case where the last byte of the pattern does not occur and the skip table is used to minimise the data analysed, and $O(nm)$ in its worst case where the pattern begins and ends with the same byte and each byte read matches the last byte of the pattern.

Despite its popularity in current DF tools, this research also investigates whether the BM algorithm remains an optimal choice for string searching within DF. The BM algorithm has been recognised as being efficient when searching for a single pattern (Skrbina and Stojanovski 2012), however, as DF investigations quite commonly require the ability to search for multiple patterns simultaneously, its effectiveness is degraded—even when modified to handle multiple pattern searches. In this regard, it is envisioned that an algorithm built specifically to find multiple patterns would be better suited for the requirements of DF investigations, such as the AC algorithm.

The AC algorithm searches with the aid of a tree topology state machine, which can search for multiple strings with a single read of the data. This state machine has two transition states. The first being a successful transition upon the next character read being part of a pattern being searched for. The second transition is a failure transition, which, depending on the character sequence read, will look for another pattern with the data processed so far. The state machine will continue to search through data until a pattern has been completely matched and location recorded, or until it reads a character which has no state. In these instances, it will reset the state machine back to its initial state and continue to read through the data stream for further patterns. The AC algorithm can match all searched patterns in $O(n)$ time for processing a data stream of length n. The AC algorithm is not dependent of pattern length m as it uses a state transition table to find all possible patterns within a single read of the data.

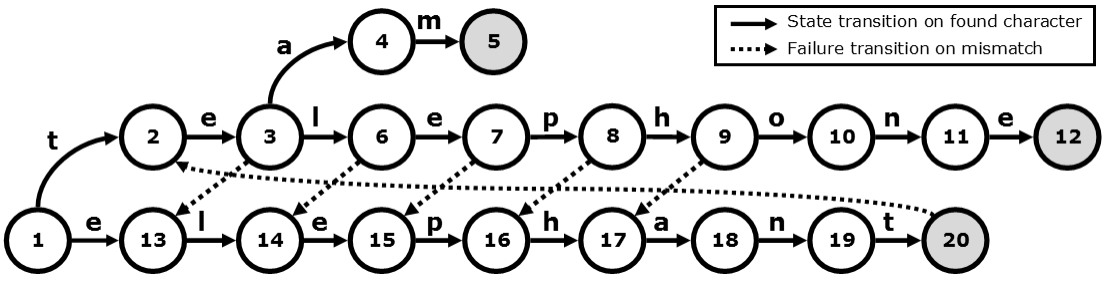


Figure 13: AC algorithm state machine example

An example of an AC state machine is outlined in figure 13 where the patterns being searched for are “team”, “telephone”, and “elephant”. If, for example, the state machine has processed up to state 9 – indicating that it has found “teleph” so far – and the next character read is an “a”, the algorithm will do a failure transition turning the active search from “telephone” to “elephant”, recognising that the data read thus far could still form part of a pattern. The advantage of the AC algorithm is its ability to search for multiple patterns in a single read of data. However, in its unaltered form, the algorithm is best suited for linear operation as the state machine cannot distribute search operations easily to multiple processing threads.

Modern parallel algorithms have flourished in the last 5 years, in which many studies have presented modified parallel algorithms that have demonstrated significant improvements over parallelising earlier algorithms (Tran et al. 2013, p. 1143–1152; Jeong et al. 2014, p. 265–272; Tran et al. 2012, p. 432–438). One of which, the PFAC algorithm (Lin et al. 2010, p. 1–5; Lin, Liu and Chang 2011, p. 1–5; Takahashi and Inoue 2012, p. 242–246) makes two fundamental changes to the way that the AC algorithm operates; firstly, by removing the use of a failure table that checks for other matches in processed data, and secondly, by requiring each byte of data to be processed individually by a separate processing thread. Whilst creating a thread for each byte of data read may seem a computationally expensive operation, if the first byte does not match what it expects, it terminates immediately, freeing the thread at an early stage (Lin et al. 2013, p. 1906–1916; Tran et al. 2012, p. 432–438). Each thread of the PFAC algorithm can search through data in the best time of $O(1)$ where the byte read does not match first byte of the searched patterns, and the worst time of $O(m)$ when the longest pattern m is matched.

Research presents the benefits of employing the PFAC algorithm, showing that the algorithm is effective at processing significant amounts of data on GPGPU devices, however, for smaller data sets, employing CPU processing may still prove more efficient (Thambawita, Ragel and Elkaduwe 2014, p. 1–4). Since its inception, the PFAC algorithm

has had many proposed changes from later research, research have suggested changes to the way data structures are allocated to better fit GPU architecture (Soroushnia et al. 2014, p. 153–160; Acharya 2014, p. 21–24) and also proposing segmented approaches to the PFAC algorithm (Agarwal, Rasool and Khare 2013, p. 52–58).

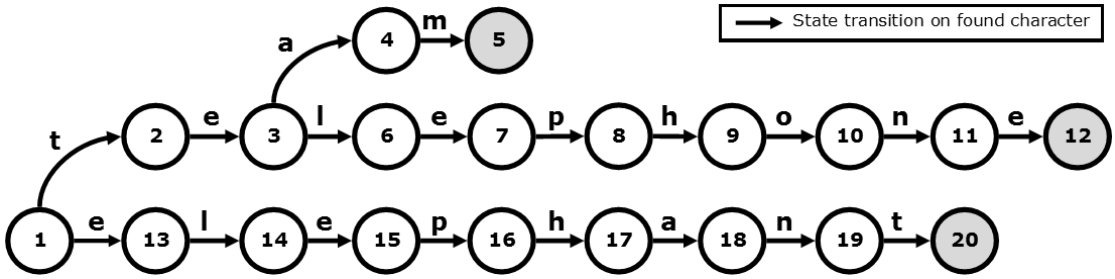


Figure 14: Parallel Failureless Aho-Corasick algorithm state machine example

The PFAC algorithm operates in a relatively similar fashion to the AC algorithm by relying on a tree-topology state machine. It differs by removing the failure transition operation from the algorithm and by introducing the requirement to process each byte of data from the tree's initial state. These changes make the AC algorithm far more usable for parallel operation as each thread processes data asynchronously, and does not require data from other processing threads. Although it is envisioned that the PFAC algorithm was designed with GPUs in mind, due to its broad array of independent processing cores, it is anticipated that multi-cored processors could also benefit from the application of PFAC algorithm.

This research has applied the PFAC algorithm to DF investigation, investigating whether this algorithm would offer substantial performance gains compared to the modified BM algorithm that is employed in Foremost. The author predicted that performance would be enhanced due to the PFAC algorithm being purposely designed for multi-pattern searching; however, it is also recognised that running a parallel algorithm in a linear fashion on a single-threaded CPU may not be as efficient as other string searching algorithms available.

3.3 Testing strategy

The following section focuses on outlining what testing strategy was used for the presented case studies. This section presents the test parameters, system setup strategy, and finally outlines the specifications of the test platforms used. Test parameters outline the purpose of the test performed and presents, and discusses, a list of controlled and free variables. System setup strategy specifies the environment set up on each test platform and what controls were put in place to minimise interference to conducted tests from external processes.

3.3.1 Test parameters

The objective of each case study was to measure what possible performance enhancements that could be achieved by introducing various processing techniques and technologies to perform string searching. A standardised test was designed for this research to gather unbiased data for each solution trialled. This standardised test involved performing string searching or file carving on forensic data – using either CPU or GPU processing – for a range of different file types by file headers.

At the end of each test, totals of how many file types were detected on the forensic data were presented. As it is known that string searching is the most computationally intensive task involved in performing file carving, no files were reproduced in string searching case studies. This allowed the tests to measure performance derived from each string searching method used. To test for scalability, each test of the case study would be tasked with finding increasing amounts of file headers; which, in turn, increased the amount of processing involved for each test. The file headers that were searched for in each test are presented in Table 1.

Table 1: File type headers

File Type	File Header (bytes)	Patterns used		
		5	19	40
jpg	FF D8 FF E0 00 10	●	●	●
jpg	FF D8 FF E1 35 FE	●	●	●
gif	47 49 46 38 39 61	●	●	●
gif	47 49 46 38 37 61	●	●	●
png	89 50 4E 47 0D 0A 1A 0A	●	●	●
tiff	49 49 2A 00		●	●
tiff	4D 4D 00 2A		●	●
wim	4D 53 57 49 4D			●
mpg	00 00 01 BA		●	●
mpg	00 00 01 B3		●	●
mp4	00 00 00 14 66 74 79 70 69 73 6F 6D			●
mp4	00 00 00 18 66 74 79 70 33 67 70 35			●
mp4	00 00 00 1C 66 74 79 70 4D 53 4E 56 01 29 00 46 4D 53 4E 56 6D 70 34 32			●
mov	00 00 00 14 66 74 79 70 71 74 20 20			●
m4v	00 00 00 18 66 74 79 70 6D 70 34 32			●
wmv	30 26 B2 75 8E 66 CF 11 A6 D9 00 AA 00 62 CE 6C		●	●
mkv	1A 45 DF A3 93 42 82 88 6D 61 74 72 6F 73 6B 61			●
wma	30 26 B2 75		●	●
m4a	00 00 00 20 66 74 79 70 4D 34 41 20			●
doc	D0 CF 11 E0 A1 B1		●	●
docx	50 4B 03 04 14 00 06 00		●	●
pdf	25 50 44 46		●	●
zip	50 4B 03 04		●	●
zip	50 4B 05 06		●	●
zip	50 4B 07 08		●	●
zip	50 4B 03 04 14 00 01 00 63 00 00 00 00 00			●
rar	52 61 72 21 1A 07 00		●	●
rar	52 61 72 21 1A 07 01 00		●	●
xar	78 61 72 21			●
xz	FD 37 7A 58 5A 00			●
jar	4A 41 52 43 53 00			●
jar	5F 27 A8 89			●
iso	43 44 30 30 31			●
cso	43 49 53 4F			●
img	50 49 43 54 00 08			●
img	51 46 49 FB			●
img	53 43 4D 49			●
cas	5F 43 41 53 45 5F			●
rpm	ED AB EE DB			●
mof	FF FE 23 00 6C 00 69 00 6E 00 65 00 20 00 31 00			●

During the design of these tests, controlled and free variables were identified to aid comparison between case studies. Controlled variables of each of the case studies were; the size and content of the forensic data used for analysis, the specified patterns searched in each test, and the test platforms used for analysis. These variables were

deemed important to remain constant across tests to compare performance between the different case studies.

Free variables were identified as possible areas where searching performance could be improved. Those include; the processing technique used, the algorithm employed to perform string searching, and the method that data is read from the storage device. These identified variables were changed in each of the OpenForensics case studies presented. Each case study discusses changes to the free variables and presents reasoning behind the changes as part of the methodology.

Revisiting the research questions, two key elements required consideration during the testing phase to determine how successful the hypothesis is. The first, and most prominent, is the factor of false-negative result accuracy. As DF is a scientific discipline, results are required to be accurate and reproducible. To facilitate this, this research used a forensic data image of an external hard drive where a number of files present in the image were already known. This was further validated through analysing the results gathered by Foremost. As Foremost is a recognised and established tool within DF, the forensic soundness of the results derived from the tool is assumed to be accurate, serving as an additional benchmark of forensic soundness.

The testing was conducted against a 20GB forensic data file that was produced from an external storage device. The external storage device was securely wiped to erase any previous traces of data on the drive and reset the contents to zero bytes. The external storage device was then loaded with a wealth of various file formats— including images, videos, audio, documents and compressed files. Each file-type consisted of the same data. The external storage device was cloned using the dd command in Linux to produce a forensically sound image. The forensic image was verified with the original storage device through comparing the MD5 checksum of the original drive and the produced image.

The methodology behind creating the forensic image for testing was not to simulate a realistic scenario, but rather to know the ground truth of how many files of

each format were contained in the image used. The files loaded on the 20GB drive also exhausted the space available, leaving little unused space on the drive. Whilst the data on the storage device is not deemed to be a realistic case, the tests performed within this research was interested in the comparative performance between the proposed and existing processing methods. It is assumed that the observed performance differences when performing string searching or file carving operations on the simulated forensic data would not vary significantly when tasked with different data.

To aid testing, baseline performance data was gathered by string searching with Foremost on the 20GB forensic image. Generated reports from Foremost produces two key values— the time that Foremost started analysis and the time that the analysis completed. From the two times produced by Foremost, the overall time in seconds analyse took and the data processing rate can be calculated, which will be later used for comparison against the proposed solutions in this research.

The OpenForensics platform which this research used reported back on all the performance metrics needed after each test. This included the time started, time concluded, the total time taken in seconds, average processing rate, total bytes analysed and patterns found. Each of the 3 series of patterns was searched for 5 times each, in which it was observed that each of the 5 times produced shown a minimal variation of less than 5%. Due to the consistency of times produced, the mean average time was used for analysis with 95% confidence levels. Result data produced from all the tests were then compiled into spreadsheets, and raw logs kept for reference. Performance speedup (S) will be calculated by $S = \frac{a}{b}$, where a is the first sample time recorded, and b is the time achieved by the second sample.

3.3.2 System setup strategy

Procedures throughout followed strict guidelines to ensure that each case study was undertaken with the same environmental variables on each test system. This ensured

that the data gathered was a fair representation of the possible performance with each test case, but that each evaluated solution could be cross-analysed for performance gain.

Except gathering the base performance metrics with Foremost (version 1.5.7) – which used Ubuntu Linux 15.10 – each system tested ran Windows 10 to test the various solutions, which had the latest updates and same up-to-date drivers installed at the time of testing— mid-January 2016. The operating system (OS) were limited to run only essential services to ensure no other third-party programs or services could interfere with the achievable performance of the solution.

The effects of caching were eliminated by rebooting the system prior to running each test case. When performing string searching with a small 5.36 GiB forensic image in earlier experiments, it was found that clearing Windows cache and performing a system memory clean was not sufficient enough to ensure repeatable times between tests. This was due to other hardware, from the storage devices used to the GPUs, caching the test's forensic data in other areas of volatile memory. When forensic data is read from cache, the tests performed would complete significantly quicker than when reading data from a storage device. It was found that caching effects were minimised by rebooting the system in-between running the same test case.

3.4 Test platforms

This main corpus of this research was benchmarked on equipment which was available to the experimenter— two desktops and a laptop of mid- to high-end specification. Table 2 shows system specifications of the computers which served as test platforms along with their allocated platform identifier. It was predicted that, despite the varying specifications of hardware, correlations would be seen between each system when comparing performance gain of the tested solutions. However, by including three

separate systems of varying hardware, we could analyse any performance bottleneck imposed by the storage device during testing.

In previous research, Zha and Sahni (2011b, p. 141–158) concluded that DF processing was disk-bound. This research anticipated that by equipping two computers with storage devices with a relatively high data transfer speed would allow far greater opportunity to analyse to what extent the storage devices limit performance. Sequential read speeds were measured for each test system using CrystalBenchMark (CrystalMark n.d.), freeware software which has a good reputation amongst technological editorial sites to accurately measure storage drive performance. The author hypothesises that the sequential read speeds of a storage device will be the theoretical maximum that forensic data can be processed at.

Table 2: Test platform specifications

Test Platform	A	B	C
Computer Type	Desktop	Desktop	Laptop
Operating System	Windows 10	Windows 10	Windows 10
Processor	Intel Core i7-5820K	Intel Core i5-4690K	Intel Core i7-4700HQ
Processor Specifications	6 Core @ 3.8GHz, 12 Threads	4 Cores @ 3.9GHz, 4 Threads	4 Cores @ 3.5GHz, 8 Threads
Processor IGP	---	Intel HD4600 (20 Core @ 350MHz)	Intel HD4600 (20 Core @ 400MHz)
Memory	16GB DDR4 2400MHz	16GB DDR3 1600MHz	16GB DDR3 1866MHz
GPU	Nvidia 980Ti (6GB), Nvidia 750Ti (2GB)	Nvidia 980 GTX (3GB GDDR5)	Nvidia 970M GTX (6GB GDDR5)
GPU Specifications	2816 @ 1279MHz, 640 @ 1255MHz	2048 @ 1304MHz	1280 @ 924MHz
Storage Device	2x 250GB Samsung Evo 850 SATA3 SSD (RAID0)	120GB Corsair Force 3 SATA3 SSD	3x 256GB Plextor M5M mSATA SSD (RAID0)
Sequential Read Performance	947 MiB/s	254 MiB/s	1305 MiB/s

3.5 Chapter summary

In this chapter, the research presents the approach taken to address the processing problems faced in DF investigation. The study developed a software platform –

OpenForensics – where different processing methods were trialled. The technologies used to create OpenForensics are stated alongside their role in processing forensic data. Consideration of the algorithms used in this research was presented. It was decided to employ a; brute-force, Boyer-Moore and PFAC algorithm to undertake string searching, and measure how quickly the selected algorithms would perform searching in the context of DF. This section further defined how testing was conducted, including details of how testing was performed on CPU and GPU implementations in the following case studies. The chapter also outlined the forensic data and the searched patterns that each test used to measure performance. Concluding, details of the three test platforms were described, including the hardware configuration, operating system, and drivers were presented.

The following chapter presents the case studies undertaken as part of this research. The case studies are presented uniformly, with an evaluation of the processing method, results from testing, and concluding with a discussion analysing the results.

Chapter 4: EVALUATION

4.1 Evaluation introduction

The aim of the evaluation presented in this thesis attempts to answer the research aim posed— to establish whether the application of GPGPU technologies and modern parallelisable algorithms could aid the problem of file carving in DF. The evaluation presents the initial base performance metric results gathered using Foremost, followed by 4 case studies with OpenForensics that introduce changes to processing approach adopted by Foremost. The final case study, case study 5, presents the developed string searching processing model to the problem of conducting file carving. Times to conduct file carving with OpenForensics will be compared to the performance derived from Foremost to measure how successful the developed processing framework is.

The evaluation will attempt to present evidence that would support or refute the research questions presented as part of the research aim. OpenForensics case study 1 to 5 presents data relevant to answer whether an OpenCL GPGPU framework provides a reliable foundation to analyse digital evidence and decrease the time required for processing forensic images without affecting accuracy. OpenForensics case studies 3 to 5 investigate whether further performance could be gained through employing a multi-string search algorithm to perform string searching with the proposed processing techniques. Finally, evidence to answer whether the potential processing rate in performing data analysis within the context of digital forensics limited by the speed of the storage device or the speed of the processor can be demonstrated from OpenForensics case study 3 to 5.

Each case study will be structured alike, presenting an introduction, aim, method, results, and conclusions of the experiment. At the end of the evaluation section, a discussion will summarise the significant findings from each case study.

4.2 Foremost: gathering base performance metrics

4.2.1 Introduction

Foremost was chosen to gather base performance metrics due to the software being open-source and widely used. As the code for Foremost is freely available to review, it was possible for this research to tune OpenForensics to closely mimic the same processing methods that Foremost employs to search through forensic data.

Base performance metrics were firstly gathered by running the file searches through Foremost, an open-source file carver still currently used today by DF professionals to perform file carving. The time Foremost takes to analyse forensic data is intended to be a fair representation of the current state of DF tools, and will later be used to base any perceived performance increases produced by the research.

Whilst it is acknowledged that baseline comparisons could have been done with Scalpel, another established file carving tool based from Foremost, earlier trials performing string searching with both tools resulted in similar times being produced with little or no significance. At the time of this research, Scalpel's GPU extension developed by Marzielle, Richard and Roussev (2007, p. 73–81) was not openly available for comparison. A more thorough comparison of how OpenForensics compares with Scalpel to conduct file carving is planned as part of future work.

4.2.2 Aim

The aim of this case study is to gather a baseline performance from Foremost to perform string searching. It is projected that the baseline performance results could be used to draw comparisons to the single-threaded CPU approaches of OpenForensics. It is anticipated that an insight can be gained on how optimised the OpenForensics

processing approach is by comparing the base performance metrics supplied by Foremost to the single-threaded CPU approaches of OpenForensics in each case study.

4.2.3 Method

As the previous chapter lightly touched upon, Foremost ran within a fresh Ubuntu 15.10 OS. The OS was live booted from an external USB 3.0 USB flash drive, where OS files are loaded and ran directly from system memory. The system memory available in each of our test systems used was deemed more than sufficient to handle both the OS and any forensic data loaded into memory. The forensic data for this test was read from the same drive as what would later be used for testing each case study presented in this research. Foremost was configured for testing for varying amounts of search patterns which were stated in a custom configuration file, and instructed only to write the audit file back to the storage drive used to read the forensic data from.

The command carried out is presented in figure 15, whilst the full configuration files used for each test can be found in appendices B1, B2, and B3. The "*-w"* flag of the command specifies that only a log file of results should be produced and that Foremost should not reconstruct files found within forensic data. It is acknowledged that, whilst albeit no files are reproduced, Foremost may still opt to conduct a second pass through data to verify file integrity. If so, the second pass may affect times produced to conduct string searching with Foremost.

```
foremost -i TestImage.dd -c /cdrom/foremost/foremost.conf -o ./foremost -w
```

Figure 15: Foremost command used with launch options

Foremost analyses forensic data in 100 MiB segments in a linear fashion and by using only a single processing thread. Although the results produced are deemed

precise, Foremost does not scale well with processor resources. It is expected that Foremost would produce modest search times in our tests. It is also envisioned that times taken to search forensic data may scale significantly with the addition of more search targets due to Foremost's algorithm choice.

4.2.4 Results

Results from Foremost of the time required to search for varying amounts of file headers are presented in table 3. Search times produced by Foremost confirm the earlier prediction that Foremost struggles to handle the additional search targets as we see each test system's search slowing significantly between the 5, 19 and 40 search target trials.

Table 3: Foremost search time results

5 defined patterns — Time (secs.)

Test Platform	Single CPU
A	114
B	160
C	104

19 defined patterns — Time (secs.)

Test Platform	Single CPU
A	415
B	453
C	440

40 defined patterns — Time (secs.)

Test Platform	Single CPU
A	741
B	761
C	792

From the results, it can be observed that all three test systems produced somewhat similar results between one another; surprisingly, however, while test platform C performed the best for searching for 5 targets, it produced the slowest times when tasked with 40 search targets. The variance in result could have been caused by Intel CPU's dynamic overclocking ability as well as performance throttling occurring due to the laptop's thermals levels during heavy processing, both of which are outside of the control of the experimenter.

Similarly, further analysing the processing rate drawn from the three test systems in figure 16 help visualise Foremost's performance obtained from the test platforms. Analysing the processing rate which each system processed the forensic data produced yet more surprising results, as none of the systems tested could process

forensic data particularly fast. Excluding test platform B's result when searching for 5 targets, all the systems produced comparable processing rates. The theoretical maximum processing rate, based upon the sequential read performance of the storage devices used, were in all cases much faster than the processing rate achieved with Foremost.

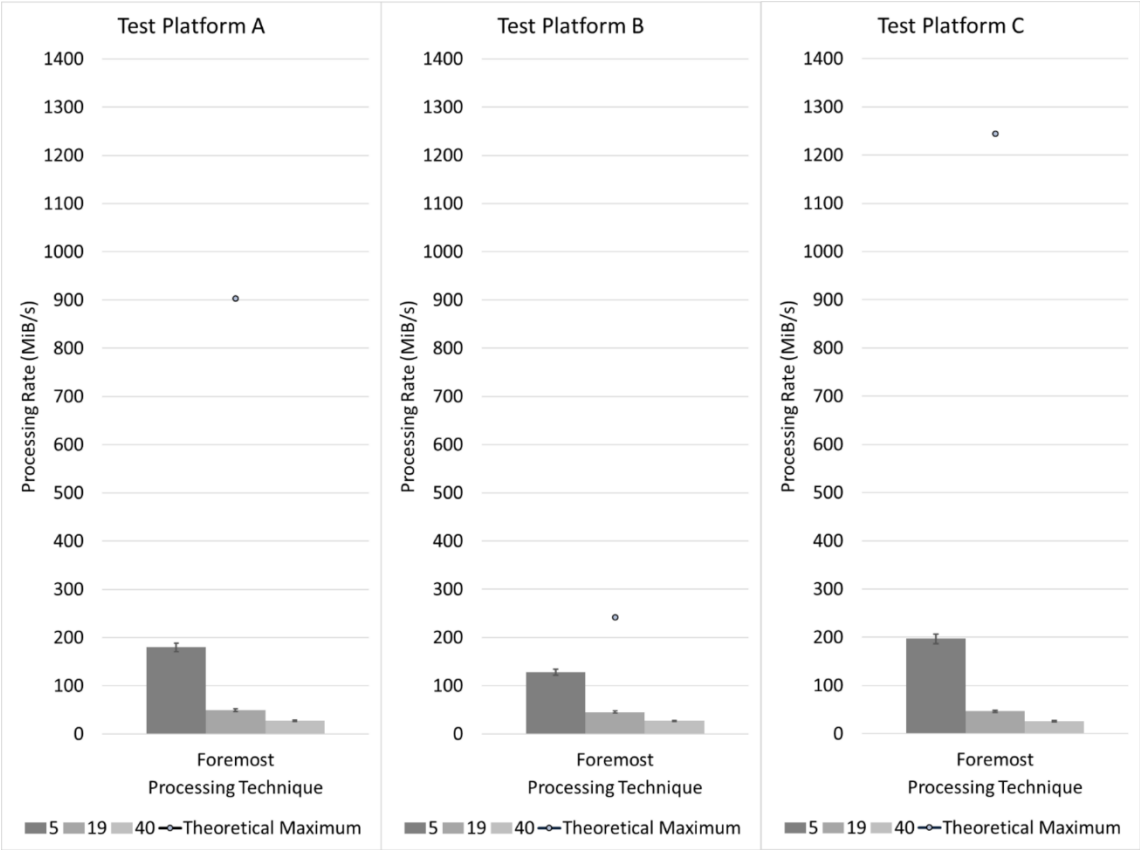


Figure 16: Foremost processing rate analysis with 95% confidence intervals

4.2.5 Conclusions

The base performance gathered from performing string searching with Foremost has confirmed a lot of known factors and limitations of the file carving tool. However, the results have also presented an unknown. The unknown being the large variance

between test platform B's weaker ability to search for 5 search targets, producing times that are significantly slower than both test platforms A and C. This specific test was rerun to validate the result in this case, which came back with the identical time of 160 seconds to perform string searching. Although test platform B has the slower storage device, it was theorised that the storage device of the system possessed enough data throughput to not become a factor in producing this slowdown and that the slowdown may have been a factor from elsewhere. However, at this stage of the research, the reason for the slowdown remained unclear as analysing the two other results obtained from test platform B shown comparable times that collated with results gathered from the other two test platforms.

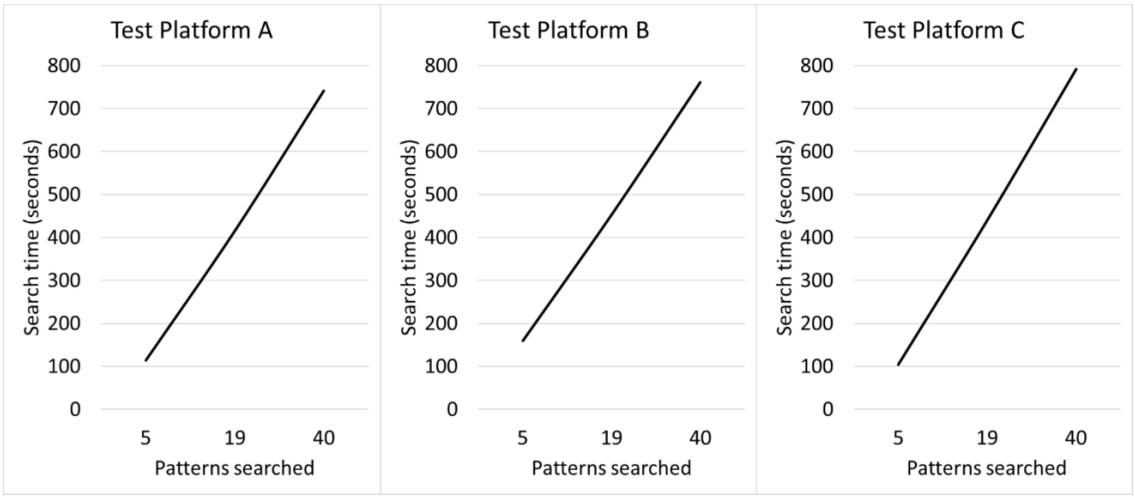


Figure 17: Foremost patterns searched and time relationship

Figure 17 analyses the time variance between searching for 5 and 19 file headers, and 19 and 40 file headers for all test platforms. It can be observed from these graphs that a clear linear trend occurs between the amount of targets searched for and the time required to complete the search. This trend signifies not only the inability of performing string searching on a single threaded CPU, but also highlights probable

inadequacies searching for multiple patterns with the modified BM algorithm employed by Foremost.

To digress, the first proposed solution that this research presents to improve upon Foremost introduces how a multi-threaded GPGPU device would tackle the problem. The research will achieve this while keeping data processing methods as close as possible to the methods that Foremost employs.

4.3 Case study 1: Using GPUs to conduct string searching

4.3.1 Introduction

The first solution presented introduces two deviations to the Foremost formula. This case study investigates the possible benefits that these changes will make to the overall string searching performance. The first change introduces GPU processing to undertake the processing associated with string searching. The GPU, in this case study, will adopt a naïve algorithm for searching through the data for patterns. The second change introduces a change to the processing cycle adopted. Currently, Foremost employs a proactive approach for checking for partial patterns split between two sections by overlapping data read by the maximum file size. We propose, as part of this case study, a reactive processing method that rewinds data only when a partial match is detected.

4.3.2 Aim

It was hypothesised that introducing GPGPUs will somewhat improve the performance of string searching through forensic data when compared to CPU processing. Even with an unsophisticated algorithm, the GPU processing technique was envisioned to surpass the performance achievable with CPU processing, a novel prediction based upon the greater processing capacities of GPUs when applying simple operations to big data. It was also predicted that the relationship between the patterns searched for and the time required to search would be less on the GPU than the CPU, due to the GPU being able to handle more simultaneous processes on its massively parallel architecture.

4.3.3 Method

The first implementation – being the focus of this case study – is how performance would be affected by introducing a GPU to perform string searching of forensic data.

The GPU algorithm adopted varies considerably from the modified BM algorithm employed by Foremost, as the modified BM algorithm is not well optimised for parallel processors working from the same forensic data. With multiple processors working on the same data, each byte of the data is distributed in turn to an available processor to process and return the result. The problem with assigning an algorithm with unique byte-skipping operations, such as the skip table of the BM algorithm, means that the processors would have to synchronise after each process to find out how far forward the next possible match lies. Synchronising a GPU is a somewhat timely operation and would likely waste valuable processing time, whereas processors held by a synchronisation request could have continued to process more data.

```
Declare int for GPU position in data
Declare int for GPU stride in data

Allocate temporary GPU memory to store results

For each byte in data segment
	If the byte is equal to the first character of the pattern
		Set the pattern is found
		If first header byte is within the last (header length -1) bytes of data
			Set rewind flag

		For each byte next to found header
			Check byte against expected pattern byte
				If byte doesn't match
					Set the pattern is not found

		If pattern is found
			Record location of first header byte

	Go to next byte

Synchronise GPU threads
Count headers found
```

Figure 18: Case study 1 GPU brute force algorithm pseudocode

Ultimately, when taking into these points, applying Foremost's modified BM algorithm for GPU processing would not make much sense and would hold back the potential processing power that GPUs have on offer. Pseudocode of the algorithm

designed to process data on the GPU can be seen in figure 18. The algorithm itself is a brute force searching algorithm that searches data sequentially start to finish. When data is loaded to the GPU, the program launches an examination on the GPU for each pattern searched for. The algorithm instructs to inspect each byte of data in a forward direction, recording the locations of found patterns within the forensic data in an empty array with a unique file type indicator. When finished with a segment, the GPU transfers the array with all the locations of found files and a found match count back to the host computer. When this data has been transferred, the CPU first checks if the beginning of a pattern was found at the end of the segment – rewinding the data back if necessary – then proceeds to process the results whilst the GPU is tasked with analysing the next segment. Checking for a partial match at the end of a segment is an operation which is not required when processing with backwards searching BM algorithm.

In contrast to the algorithm devised for GPU processing, the CPU algorithm employed the same processing steps used by the modified BM algorithm seen in Foremost. The CPU algorithm pseudocode used to process forensic data is presented in figure 19. In early experiments before conducting the first case study, the research experimented both with creating a skip table from a combination of the patterns searched for and creating a skip table for each pattern searched for. Results from these earlier experiments revealed that searching was conducted faster when each pattern was searched for individually rather than in combination. The concluding reason for this result was that with so many different patterns being searched for, the skip table became less and less effective. When searching for multiple strings, the skip table became more-or-less as efficient as a byte-by-byte brute-force search. Searching each pattern individually in memory, however, took full advantage of BM's skip table to search for each target byte, proving surprisingly more efficient for the CPU workload in trials.

```
For each pattern being searched for
Create a counter for position in data initialising at (pattern length -1)

	For each byte in data segment
		If the current byte is equal to the last byte of the pattern
			Set the pattern is found

			For each byte before the found byte
				Check byte against expected pattern byte
					If byte doesn't match
						Set the pattern is not found

			If pattern is found
				Record location of first header byte

			Go to next byte

		Else
			Go to next byte using skip table
```

Figure 19: Case study 1 CPU modified Boyer-Moore algorithm pseudocode

The second proposed variation introduced a change on processing cycle and how data is searched. Early research from this case study examined how Foremost processed data, and a different approach was adopted in this study to attempt to create a more efficient way of searching for files that may occur at the end of sections. This check is used to ensure that when a file header is found near an end of a section without matching file footer, the program will react and rewind its position in the forensic data to ensure a file has not been split into two data sections. While this is done in Foremost with the use of a windowed technique, by overlapping all segments of data by the maximum file size being searched for, the research proposes only to rewind the position in the forensic data when a partial match has been found— creating a reactive rather than a proactive response.

While the tests in our case study are only interested in the search for the headers of files and not complete files, this change is only used in the tests by the forward searching GPU algorithm in ensuring file header itself is not split between two data sections. The reverse searching modified BM algorithm employed in this study uses a

reverse search method, signifying that any file header matches cannot be split between two sections. If the program detected the start of a pattern which was searched for, the program would flag for the program to rewind the data back a number of bytes to account for any patterns which may exist between the two sections. A diagram that outlines the full revised processing cycle is presented in figure 20, which both the CPU and GPGPU processes adhere to. The dotted arrow lines of the diagram signify the check which the GPGPU algorithm uses to check for an incomplete file header.

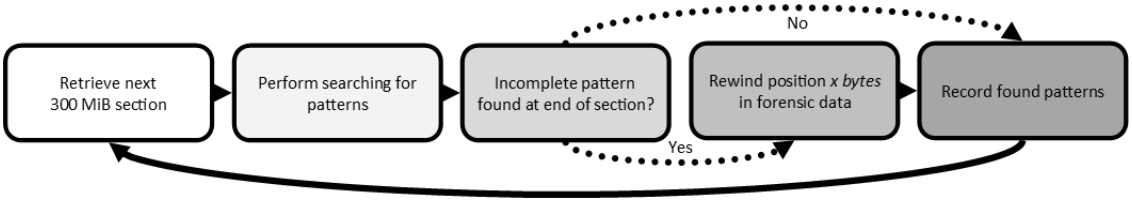


Figure 20: Case study 1 processing cycle

The final proposed variation is the size of the segments that are processed, increasing the size from 100 MiB to 300 MiB. This change was made to test the theory on whether the benefit of reducing the number of times required to check for patterns or files which may be split over two segments would outweigh the timely operation of pre-loading greater amounts of data from storage device to memory.

4.3.4 Results

The times taken to search the forensic data are presented in table 4. The results presented some intriguing findings. The times taken to search the forensic data using the proposed modified BM algorithm on the single CPU in this case study are far greater than times produced by Foremost. This indicates that the changes made to the algorithm, primarily instructing the CPU to search for each pattern separately rather than in combination, slowed searching down significantly despite earlier experimentation. Despite the disappointing performance from the single CPU

implementation derived from this study, comparisons can still be made between the GPU and IGP times and the base performance results gathered with Foremost.

Table 4: Case study 1 search time results

5 defined patterns — Time (secs.)

Test Platform	Single CPU	Single GPU	Single IGP
A	222	44	48*
B	274	98	109
C	253	49	56

19 defined patterns — Time (secs.)

Test Platform	Single CPU	Single GPU	Single IGP
A	1198	50	64*
B	1237	104	158
C	1384	61	114

40 defined patterns — Time (secs.)

Test Platform	Single CPU	Single GPU	Single IGP
A	2461	58	88*
B	2471	113	232
C	2850	80	199

** - Secondary discrete GPU, no IGP present on system*

From the time results, the GPUs and IGPs from all test platforms managed to achieve respectable performances. All significantly besting times derived from the CPU implementation as well as the previous Foremost tests. Nonetheless, this was an expected novel finding, as the processors were processing data much faster on a massively parallel scale. Furthermore, the CPU implementation is limited to run only one thread, not utilising the full computational power of the processor. The fastest time to search all 40 search targets, achieved from test platform A's GPU, took only 58 seconds— a phenomenal result which surpassed initial expectations from applying a minimalistic algorithm to conduct string searching through forensic data.

When comparing the result from test platform A's GPU to the time Foremost took to search for 40 patterns (741 seconds), the GPU algorithm performed searching for 40 search patterns 12.78x faster. This result was not unusual when comparing test platforms B and C's GPU results, both which delivered 6.56x and 9.26x faster performance respectively. Slight differences in deliverable performance enhancements in these tests can be explained by the variation in processor hardware between the different test platforms, some which have more powerful processors than others.

Test platform's B and C's IGP also delivered impressive results from the time results, with the initial unexpected observation that the laptop's Intel HD4600 outperformed its desktop counterpart of the same model. With further inspection,

however, this result is explained by referring to the precise specifications of the two IGPs. The Intel HD4600 of test platform C is clocked 14.3% higher (50 MHz) than the Intel HD4600 of test platform B— resulting in faster speed in processing forensic data.

Test platform A's secondary GPU, the Nvidia 750Ti, performed commendably too, producing only marginally slower results than the high-end discrete laptop GPU found on test platform C. Nevertheless, it is also observed from the results of this case study that test platform A's secondary GPU seemed to show the most deterioration out of all discrete GPUs when tasked to search for increasing amounts of patterns.

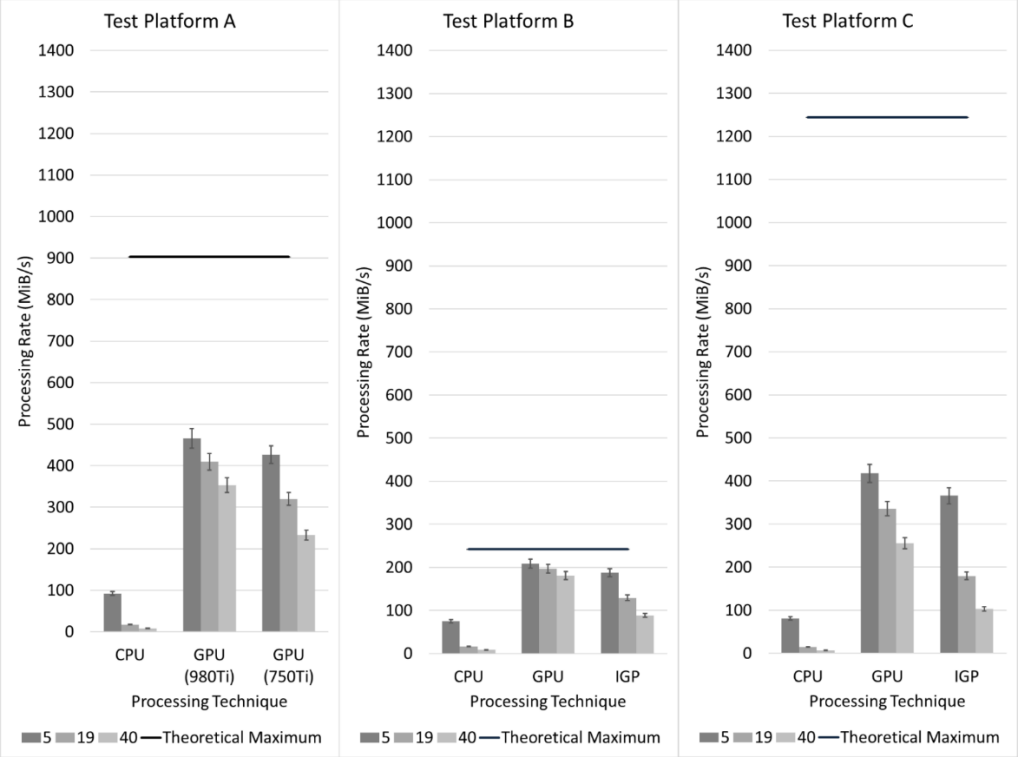


Figure 21: Case study 1 processing rate analysis with 95% confidence intervals

One of the fascinating results from this first case study is seen from the processing rate analysis as shown in figure 21. As anticipated, the quicker processing times achieved from the GPUs and IGPs on the test platforms have translated to significantly higher processing rates. However, the proposed solutions of this case study

have failed to achieve the theoretical maximum speed that the storage devices can read forensic data. The test platform with the slowest storage device on trial, test platform B, demonstrates that even by conducting searching on a GPU with a naïve algorithm, searching may not be limited by storage device transfer rates, but rather by the technique employed to search for evidence.

Also from analysing the processing rate and times gathered, it is also hinted that the GPU and IGP times seem to depreciate less when searching for larger amounts of patterns. Analysing this further, figure 22 visualises and confirms this hypothesis to be true, showing that both GPUs and IGPs demonstrate significantly less time deterioration when more search patterns are defined. This observation is due in large part – once more – to the GPU and IGP's ability to parallelise searching of multiple targets better than both the single threaded CPU implementation in this study as well as Foremost's method of searching.

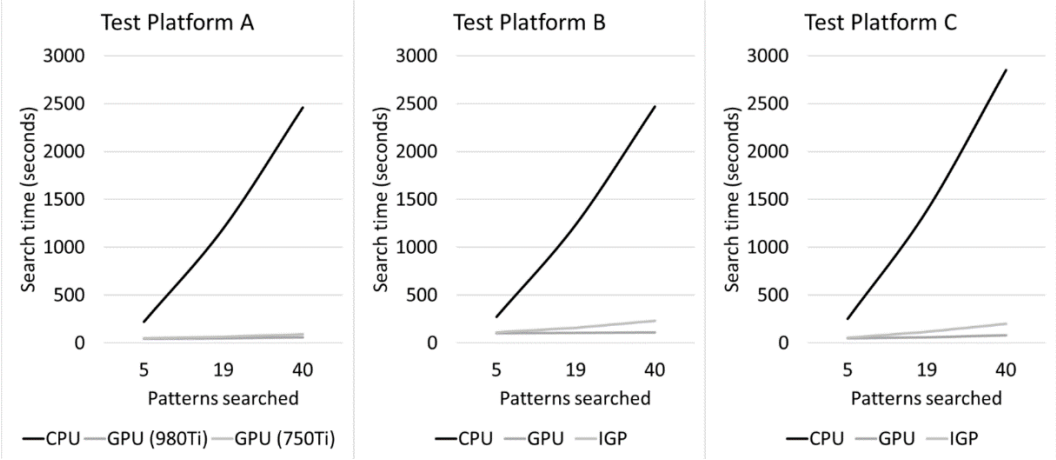


Figure 22: Case study 1 patterns searched and time analysis

4.3.5 Conclusions

This case study presented two changes to Foremost's formula to try and improve upon string searching within DF. While Foremost is limited in design to only use only a single

thread of a CPU to search through forensic data; it is found to run well despite this limitation, significantly outperforming this case study's single threaded CPU implementation. Notwithstanding this case study's CPU algorithm being designed around Foremost's algorithm, the study found that the changes made to the algorithm and processes were not as optimised as anticipated. It is granted that other factors, such as the different OSs each solution was ran within and languages each solution was developed in may have had significant effects on the resulting times. Even so, this case study highlights important lessons to take forward to optimise further the approach of the GPU and IGP approaches in conducting string searching.

Analysing the changes in this case study, the introduction of conducting string searching on GPUs proved very successful and provided significant performance increases over both the study's CPU implementation and Foremost alike. While this was expected in our initial predictions, the hypothesis was that the possible performance of the GPU implementation may have been limited by storage device data transferal rates. After further analyses of the results, however, it became apparent that processing on the GPU did not utilise the full capability of the storage device— even with the test platform with the slowest storage device on the test. IGPs found on the CPUs of test platforms B and C also performed well during testing, while slower than the discrete GPUs found on the test platforms, they proved viable processors to conducting string searching. As most modern mainstream CPUs available now in consumer and workstations are equipped with some form of IGP, it would be beneficial to utilise the power behind these capable chips to provide additional processing power for the discrete GPU— treating the IGP as a partnered asynchronous GPGPU processor. With this change, it is anticipated that searching could be performed faster still.

The result gained by test platform B's GPU raises another further peculiar result when comparing the performance shown by each platform's discrete GPUs. Platforms A and C's discrete GPUs, the Nvidia 980Ti, 750Ti and 970M, all managed to process the forensic data significantly better than test platform B's Nvidia 980 GPU. With the

specifications of the GPUs on test, it was expected that test platform B's Nvidia 980 should have attained processing rates in between the Nvidia 970M of test platform C and the Nvidia 980Ti of test platform A— signifying that the GPU on test platform B should have been able to hit the theoretical maximum processing rate for this system.

An explanation can be found when revisiting the processing cycle adopted to process the forensic evidence in this case study. With the processing cycle used, the data is not processed asynchronously by any of the processors within this case study. Instead, data is handled in a rather synchronous way, where the storage device will only fetch the next segment of forensic data when the processor has finished processing the current data segment— meaning that between processing, the storage device sits idle until instructed to serve the next segment of data. Within the results of this case study, synchronous processing can be seen to have a negative effect on performance that makes the theoretical maximum unachievable by any of the tests demonstrated— due to the storage device idling during processing.

Concluding on the introduction of GPUs, further experiments should be extrapolated and further performance enhancements introduced, such as the introduction of multiple GPU processing and more sophisticated GPGPU algorithms, to discover whether the theorised maximum processing limit of the storage device could be reached, or whether there are any other factors which may limit the speed analysing forensic data.

The second change introduced in this case study attempts to modify the processing cycle employed by Foremost by only rewinding the position in the forensic data back when an incomplete pattern is found at the end of the current section. This change seemed logical at first. However, further iterations of this research would deteriorate the possible performance with this technique, particularly when exploring multi-threaded processing. This is because if one thread – or one processor – flags the requirement to check data at the end of its current section, it may cause the storage device to make significant jumps back and forth between locations on the storage device

to serve data to processing threads. All the jumps back and forth through data would inevitably cause delays to processing forensic data, with a greater impact on traditional HDDs where a physical movement of the disk platters and read heads are required to find the data requested.

While the tests presented in this research are only interested in file headers and the searching of forensic data without extracting files, the tests carried out are less impacted by the problem above as the chance of finding patterns of a few bytes long being split between data segments is unlikely. However, in the interest to present the best possible way to explore forensic data for the purposes of reconstructing files, it is deemed that future experiments should report back to a more multi-threaded friendly way of employing a windowed technique. Akin to the technique employed by Foremost, having an overlap of the maximum possible file size between sections to ensure files are not split into two sections. By reporting to the windowed section technique, it is anticipated that storage devices will be more efficiently used when conducting string searching and file carving on forensic data on traditional HDDs.

Part of this case study increased the file data segments that the forensic data is split was into from 100 MiB to 300 MiB. It was anticipated that comparisons could have been possible between Foremost and OpenForensics CPU results; however, due to the OS, processing, and other unexpected differences, this case study supplies little evidence that supports or refutes that larger segments enable faster searching through forensic data. In the next case study, the file data segments will be reduced to 100 MiB to see if it has any impact on the times produced by the test platforms on single threaded CPU tests. It is also worth noting that when experimenting with parallel multi-threading on CPUs and multiple GPGPU devices, it is envisioned that smaller data segments would benefit systems with limited availability of main memory as the volatile memory required to perform searching in parallel would grow exponentially with the number of threads employed by the CPU, or, GPGPU devices used.

4.4 Case study 2: Utilising asynchronous parallel techniques

4.4.1 Introduction

The previous case study showed some immediate advantages when employing GPUs to conduct string searching within forensic data. With this said, there were some lessons learnt about the execution of searching that this case study aims to address, as well as obvious improvements which could have been made to further enhance searching performance.

This case study investigates how GPGPU processing compares with a fully utilised CPU to conduct string searching. This is achieved by implementing the use of threaded processing to both CPU and GPU processing. The second change introduced by this case study reverts to a Foremost style of proactive searching with the “windowed” section technique. This change, albeit reverting to a proactive search technique, was done to optimise searching on traditional mechanical storage devices. This is deliberated in more detail in the discussion.

4.4.2 Aim

The aim of this case study is to demonstrate further performance gains by performing string searching through employing multiple processors, or multi-threading, approaches. Results will be collated in the same way as presented in the last case study to ensure consistency and make presentation of performance gains easier to analyse between studies. It is also hoped that through employing a parallel multi-GPU approach, evidence of a performance limit can be witnessed when analysing data processing rates, confirming that theoretical data processing limits exist when processing forensic data.

4.4.3 Method

The most significant change that this case study presents is the use of multi-threading and multiple processors to perform searching for patterns within forensic data, using an asynchronous model where each processing thread acts independently. It is envisioned that the employment of such techniques would offer substantial benefits. However, performance may be limited in some cases by the theoretical maximum data transferal rate of storage devices that forensic data is read from. Experiments with multi-threading and multi-GPUs would indicate whether the theoretical maximum data transferal rate is bound solely by the storage device, or whether there are other confounding factors which limit performance from performing string searching within the context of DF.

To help analyse the extent of performance gains between this case study and the previous case study, the same algorithms were employed in this case study to conduct the string searching on the forensic data. The GPU algorithm utilising the same brute-force algorithm, and the CPU implementation – albeit less efficient than Foremost's execution – still employed the modified BM algorithm, as outlined in the pseudocode of figures 18 and 19 within the previous case study.

As previously mentioned within the earlier study, there was little evidence of performance gains from searching in larger sections. This case study read forensic data in 100 MiB section blocks to gain more understanding on whether the change of section size effected searching positively or not. Further, the previous case study changed the method that the program handled checking for patterns which may have been split into two sections. In the tests conducted, it was deemed highly improbable that a header pattern of several bytes would be split between two sections, however, far more probable that a whole file – which may be several MiB – may be divided between sections when searching in smaller data sections of 100 MiB.

When designing the multi-threaded approach to performing file carving within the context of DF, it was deemed that the processing method adopted in the previous

case study would have caused delays when used in a parallel scenario with multiple running search threads. Particularly when reading data from traditional HDDs. HDDs differ from modern SSDs by having mechanical parts that spin the disk platters where the data is stored, and a head assembly mounted on actuator arms that are used to read data. Reading data at different locations on the disk platters causes seek time, where the storage device endures a time delay to move the head assembly on the actuator arm to the place on the disk platters where the data is located.

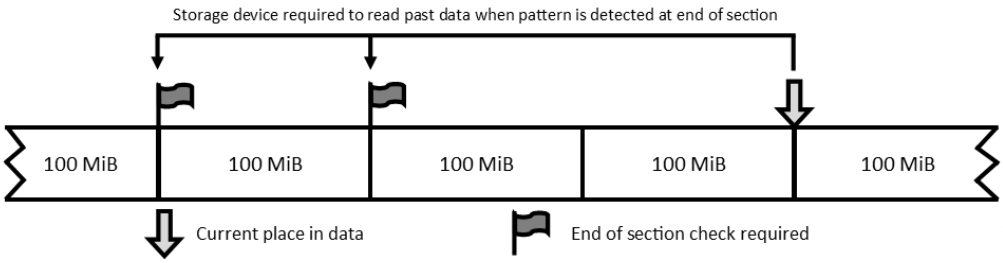


Figure 23: Case study 1 section processing approach

The first case study's processing approach, as shown in figure 23, may have caused HDDs a delay, as if multiple processing threads detected a partial match at the end of a section, the HDD would be tasked with reading previously processed data as well as fetching current data for other processing threads. The requirement to check historic data may cause additional time to read forensic data stored in different areas of the storage device. When searching in a parallel fashion, each additional processing thread could potentially mark an end of section check on the segment of data the thread has analysed, exponentially increasing the time required by the storage device to read forensic data.

The processing cycle adopted in this case study is presented in figure 24, where each processor follows the same simplistic approach to process data. In multi-threaded and multi-GPU approaches, the available processors on the system work asynchronously in parallel to handle all of the forensic data in 100 MiB segments. Processing is carried

out sequentially, where each segment only requires being processed once. In turn, each segment of forensic data is dynamically assigned and processed independently by an available processor until all of the segments of forensic data have been processed. Within the context of the string searching tests performed in this case study, discovered file header patterns are recorded in memory and presented back to the user when all forensic data has been processed. If file carving, files are reproduced when the processor finishes an analysis of a section.

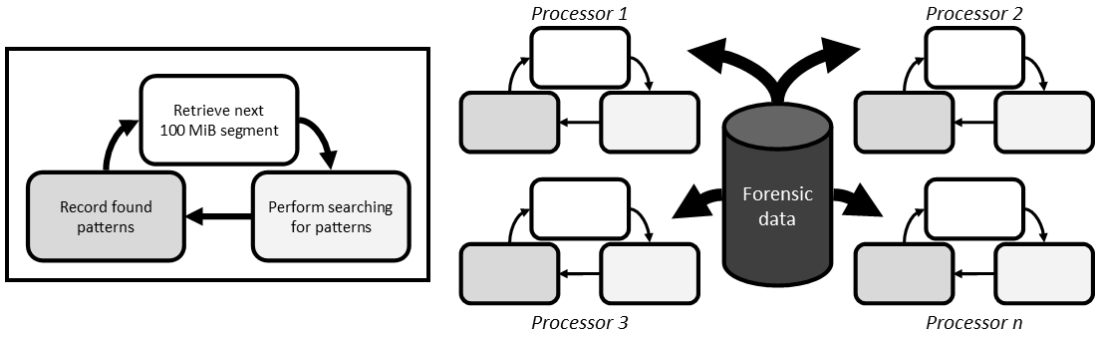


Figure 24: Case study 2 processing cycle

It was deemed logical to resort back to reading data in a windowed fashion, where data is read in sequence with an overlap of the largest possible target size specified in the configuration— for this case study, the window size was the length of the longest file header. This was foreseen to be a more optimised approach, ensuring that slower performing mechanical HDDs would not be disadvantaged when performing file carving and ensuring that forensic data is only read and accessed once, which should – in theory – have a positive effect on file carving performance.

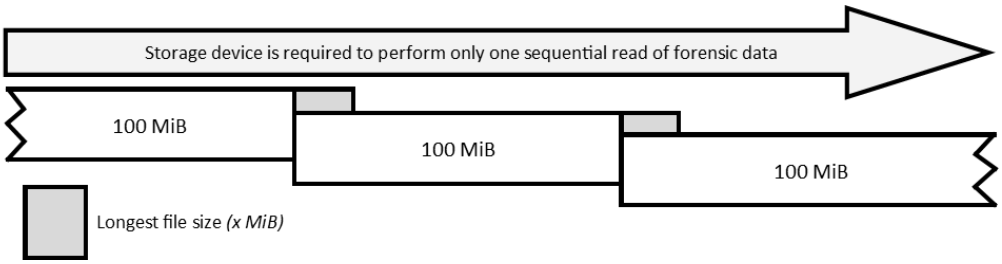


Figure 25: Case study 2 section processing approach

4.4.4 Results

The results produced from this case study are outlined in table 5. Presented are the times which each technique took to conduct string searching on the forensic data with the varying amount of search patterns defined. A few differences can be seen when comparing the times achieved by the single CPU, GPU, and IGP to the previous case study. Despite observing a few improved times, it is generally shown within the results that performance has degraded slightly within this case study; indicating that, albeit an insignificant variation, the changes to both the segment sizes that forensic data is split into, and the changes to the processing cycle may have produced a negative effect on the performance achievable with the algorithms and technology used.

Aside from the minor time variations between the two case studies, this case study presents exceptional results from applying multi-threading and multi-GPU technologies to carry out string searching on forensic data. Most notable results are provided by the most powerful system on test – test platform A – which manages to reduce the time required to search for 40 defined search patterns from 2418 seconds to 341 seconds through using all 12 available logical CPU cores on the processor to search through forensic data. Likewise, test platform A's application of both GPGPU devices reduced the time from 69 seconds from using the platform's fastest GPU to only 47 seconds to process all 20 GB of forensic data.

Table 5: Case study 2 search time results

5 defined patterns — Time (secs.)

Test Platform	Single CPU	Multi-CPU	Single GPU	Multi-GPU	Single IGP
A	220	33	43	27	48*
B	254	94	94	85	118
C	259	47	44	36	73

19 defined patterns — Time (secs.)

Test Platform	Single CPU	Multi-CPU	Single GPU	Multi-GPU	Single IGP
A	1182	168	53	34	67*
B	1234	306	104	87	170
C	1403	303	59	46	124

40 defined patterns — Time (secs.)

Test Platform	Single CPU	Multi-CPU	Single GPU	Multi-GPU	Single IGP
A	2418	341	69	47	96*
B	2444	613	120	102	245
C	2823	624	82	65	206

** - Secondary discrete GPU, no IGP present on system*

Unlike test platforms A and C, test platform B was already quite close to the theoretical maximum performance limit of the storage device used to read the forensic data within the previous case study, hindered by the synchronous processing cycle adopted for single processor testing. Times produced from test platform B's multi-GPU tests suggest that that theoretical processing rate may have been met, as the time produced by test platform B for the multi-GPU test does not show the same pattern of performance gain as observed by the other two test system's 5 pattern tests. This observation is validated when we calculate the processing rates produced by each platform in figure 26.

The processing rates which each platform produces offers further insight on how applying an asynchronous multi-GPU and multi-threaded approach affects the performance achievable. As predicted at the beginning of this study, the results show significant improvements. Multi-threading on the CPU show the most benefit over its synchronous counterpart, as the CPUs employ all logical cores to process data instead of using just a single core. The multi-GPU results also show noteworthy improvements by employing all GPGPU devices available on the test system to perform string

searching. In the case of test platform B's 5 pattern search, the multi-GPU string searching performance appears to be limited by the performance of the storage device.

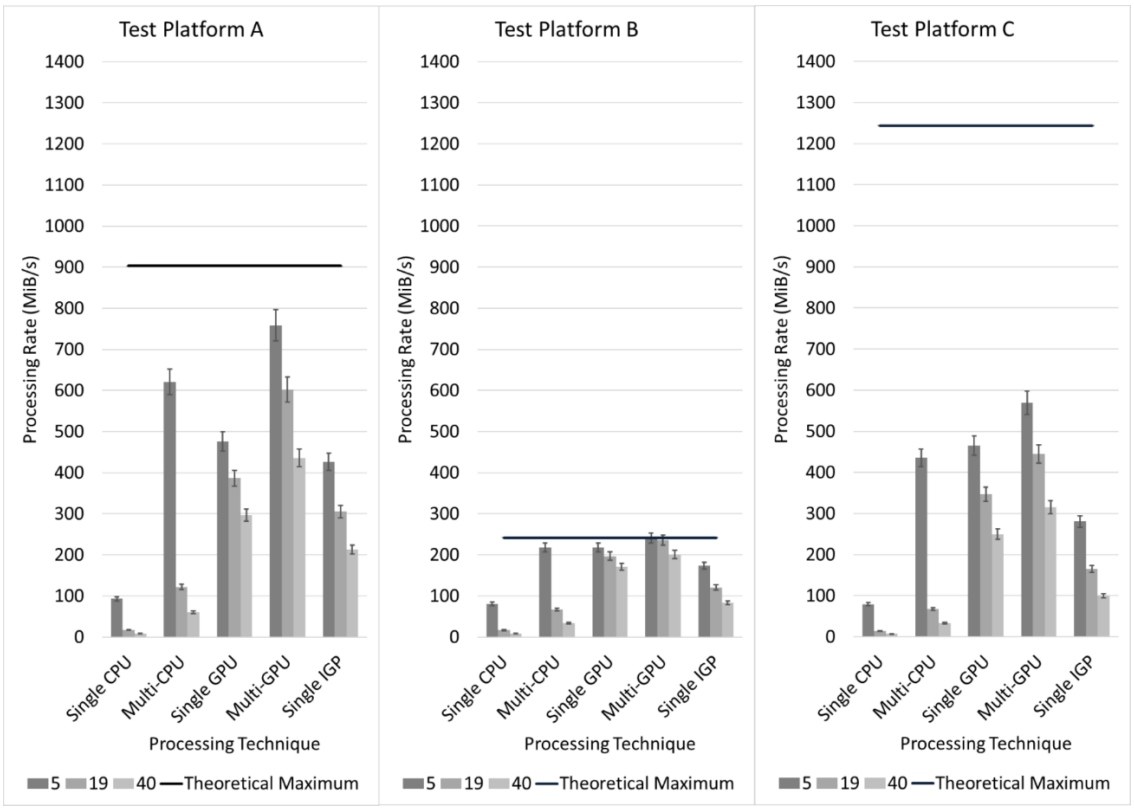


Figure 26: Case study 2 processing rate analysis with 95% confidence intervals

Multi-GPU results don't initially appear as impressive as multi-threaded CPU results mainly due to CPUs having more headroom for improvement when employed in a multi-threaded approach; for example, test platform A can spur twelve asynchronous threads – one for each logical CPU core – whilst only instructed to create two synchronous threads for the two GPGPU devices— signifying a maximum potential speedup of 12x for the CPU and 2x for the GPU. In consideration of this fact, while the multi-threaded CPU does produce much better results over its single-threaded counterpart, the performance is still relatively minor in the 40 pattern search when compared to that gained from all the GPU, IGP and multi-GPU respectively.

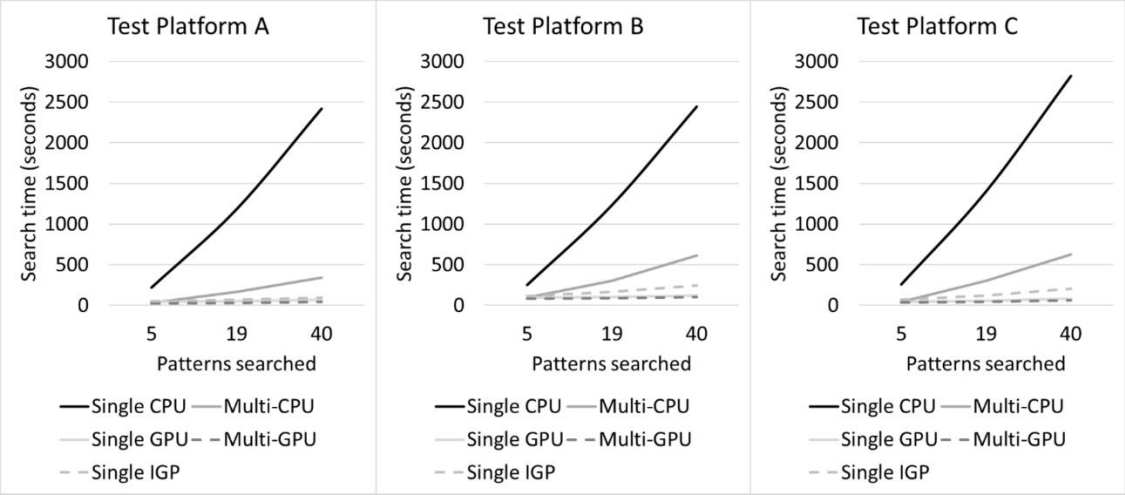


Figure 27: Case study 2 patterns searched and time analysis

When analysing the relationship between the time taken and patterns searched for in figure 27, it can be identified how each processing technique's processing time is affected when tasked to find more patterns. Within these graphs, it can be identified that utilising multi-CPU processing has an overall favourable advantage when searching for larger amounts of patterns within data, showing not only a significant reduction in time required, but also less time deterioration when tasked with to search for more patterns. Nonetheless, due to the limitations of the modified BM multiple string searching algorithm as well as the underlying processor architecture, both CPU techniques are significantly outperformed by GPU, IGP and the combination of multiple-GPGPU devices when performing searching for 19 and 40 patterns— with the latter multi-GPU solution demonstrating the best performance as predicted.

The results can be further interpreted by investigating the speedup of each technique over the single CPU technique, as shown in figure 28. Test platform A demonstrates average 6.9x speedup over all tests utilising all 12 logical cores on the CPU. Whereas test platform B that employs 4 processing threads shows an overall 3.6x speedup, and test platform C that employs 8 threads demonstrates 4.9x speedup

compared to their single-threaded counterparts. The 5 pattern result from the multi-CPU method of test platform B is seen to show only a 2.7x speedup, skewing the average result for this platform. The result from this test is seen to also perform similarly to the single GPU test, which was found in the first case study to be limited by the storage device idling when the single processor was processing forensic data, however, as the multi-CPU processes data in an asynchronous fashion between the different processing threads, it would be improbable that storage device idling would be the problem.

Further, closer inspection of the multi-CPU processing of this test confirms that the delay was due to the four processing threads employed by the CPU not being able to process data quick enough, instigating the storage device to idle when all four threads were actively processing and did require data from the storage device. It is assumed that the storage device would have less time idling if test platform B had access to more threads to process forensic data on, or alternatively if a more optimised algorithm was employed to process forensic data on the CPU.

Inspecting the speedup over the single CPU, GPGPU processing excelled over CPU techniques when searching for increasing amounts of patterns. GPGPU processing had a clear trend of the greater amount of patterns searched for, the larger the speedup would be. This can be explained, like in the previous case study, by the GPU and IGP's ability to massively parallelize simplistic mathematical problems. Technology onboard GPGPU devices possess far greater amounts of algorithmic units which make it capable of processing data at a far greater rate than CPUs. Speedups reached a significant 51.5x faster when using multi-GPUs to search for 40 patterns within forensic data on test platform A. Test platforms B and C also reached impressive speedups of 24x and 43.4x respectively when performing the same tests.

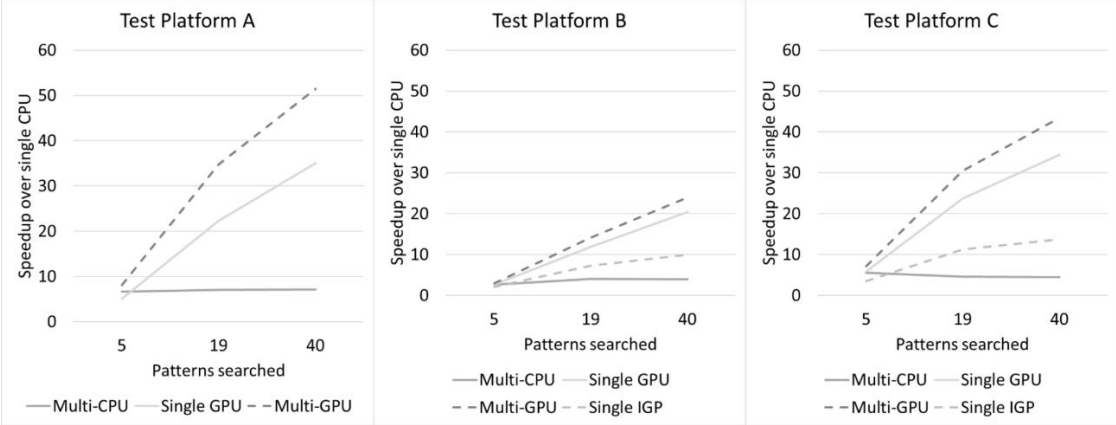


Figure 28: Case study 2 technique speedup over single CPU solution

4.4.5 Conclusions

This case study's goal was to improve upon the results produced by the last case study by introducing some variables into the existing framework. The first – and most significant – of implemented changes being the introduction of asynchronous multi-threaded CPU and multi-GPU processing to conduct string searching on forensic data. It was predicted that introducing parallelization to process forensic data could significantly improve the performance achievable to that attained by the previous case study, wherein only a single threaded, or single GPGPU, the approach was adapted to process data.

Results produced by this case study confirm this hypothesis, as adapting parallel techniques on the CPU produced a speedup of 3.6-6.9x over its single threaded operation. Likewise, applying multi-GPU techniques achieved an average speedup between 1.2-1.5x over utilising just a single GPU. The largest of the observed speedups was produced by test platform A, which employed two discrete GPUs to process data. Even though utilising parallel processing provided the CPU with far more significant speedups, CPU processing also had the greatest headroom to improve. Overall, it is recognised within this case study that partnering all available GPGPU devices on a system will produce the best performance when processing forensic data.

Processing performance limitations were observed only when performing string searching for 5 patterns with multiple GPUs on test platform B. As the test platform with the slowest storage device on trial and performance closest to the theoretical maximum in the previous case study, it was anticipated with the introduction of asynchronous processing that performance bottlenecking may have occurred for this test platform. This result provides an early indication that the theoretical maximum indeed does exist, however, one instance of this anomaly is deemed insignificant until further evidence of the bottleneck is presented.

In the case of test platform B's multi-CPU result, storage device idling could still be witnessed due to processor unavailability; this is due in large part to the lack of processing threads and/or the suspected lack of an efficient multi-string searching algorithm. While the same could be claimed by GPGPU approaches, GPGPU devices using a simple brute force algorithm still possessed enough raw processing power to process segments of forensic data in less time than it takes to read the next segment of forensic data from the storage device. The GPGPU processor's efficiency at processing data significantly minimises the amount of time that the storage device is idle, and when pairing GPGPU devices, more of the storage device's performance can be utilised. In test platform B's case when using multi-GPU processing, we can see that the theoretical maximum performance of the storage device is fully utilised.

To improve these results further, an investigation on how a parallel multi-string algorithm could reduce the time required to search for multiple patterns within forensic data. As both the CPU and GPGPU both use algorithms which are seen to be ill-suited for performing string searching for multiple patterns. It is hypothesised that significant advantages could be reaped from employing a parallel friendly multi-string searching algorithm. It is anticipated that the degradation of performance would be less in all tests conducted. However, it is expected that, similar to this case study and the last, utilising GPGPU processing will show the quickest results and greatest performance gain. It is unknown, however, how the introduction of an improved multi-string algorithm would

affect the performance gap seen between synchronous and asynchronous implementations of the CPU and GPGPU methods— whether the performance improvement gained through asynchronous parallel deployment would increase, or otherwise diminish.

Within this case study, the segments which the forensic data was separated into was decreased from 300 MiB to 100 MiB to facilitate parallel processors having enough independent memory space to process and record results asynchronously. This change was applied globally to all processing techniques to measure how the reduction in segment size affected the speed which data was processed. Results largely show a negative effect when comparing single CPU, GPU, and IGP results of this case study to that of the previous case study. Whilst the difference between the two sets of results is arguably small, it remains enough of a difference to demonstrate that slightly enhanced performance can be gained from splitting forensic data into larger 300 MiB segment sizes for processing within a single threaded application.

Within an asynchronous processing model, however, segment sizes which the forensic data is separated into must be treated differently than synchronous processing. In the asynchronous processing model adopted by this study, segment sizes are mostly limited by the amount of RAM memory available on the system to store each processor's current data segment and results. Whilst the system memory available on all three of the test platforms in this research are deemed plentiful and could entertain handling 300 MiB of data per processor, having larger data segments may also introduce processor blocking— whereas available processing threads are held idle by the storage device transferring data segments to other threads. It is predicted from these observations that segment size may be best allocated dynamically, taking into consideration the amount of processors and RAM available for the analysis and other search parameters – such as maximum potential file sizes – to adopt an optimal file segment size to separate the forensic data into without blocking either the storage device nor asynchronous processors.

Another change made in this case study was altering the processing cycle that the processors used to process forensic data, modifying how checking is done for results that may be split between two sections. The reactive detection-based approach of the previous case study was changed to a proactive approach of having a small overlap between segments. The results of the reactive approach gained from the last case study showed promising results, however, the tests performed searching for file headers likely gave the reactive processing cycle an advantage, as the chance of discovering fragmented headers of several bytes is significantly less than when searching for whole files of several MiB. Nonetheless, tests performed with the proactive processing approach shown little degradation to the overall time taken to search forensic data.

With the advantages that proactive processing brings, especially considering the benefit of reducing the seek time required from traditional HDDs when searching forensic data, it is the belief of the author that a proactive processing cycle would provide the overall quickest and most reliable file carving times when tasked with reproducing files from forensic data.

4.5 Case study 3: Employing the parallel failureless Aho-Corasick (PFAC) algorithm

4.5.1 Introduction

Within the previous OpenForensics case studies, experiments have been completed using a modified BM algorithm to perform CPU searching, and a brute-force algorithm to perform GPU searching. Case study 2 has shown significant performance gains when employing single- and multiple-GPUs to conduct string searching. This case study investigates whether further performance could be gained through employing a multi-string search algorithm to perform string searching with the proposed processing techniques.

It is anticipated that with the employment of a better multi-string algorithm that CPU and GPGPU processing could both benefit with enhanced string searching performance. With the introduction of a more optimised algorithm, however, it is expected that performance advantages in some test cases may be limited by the theoretical maximum sequential data transfer speed of each test platform's storage device.

4.5.2 Aim

This case study aims to demonstrate how each processing technique would perform with a more optimised multi-string searching algorithm – the PFAC algorithm – to perform string searching within the context of a DF investigation. This case study will compare and interpret the attained results to those produced by case study 2.

4.5.3 Method

The implementation of the PFAC algorithm into OpenForensics entailed modifying two entities; the pre-processing of searched for patterns, and the processing steps that both CPU and GPGPU devices followed. The PFAC lookup table generation is processor agnostic, in which both CPU and GPGPU implementations can follow the state machine table to look up their next instruction. The processing steps, on the other hand, are not due to the requirement of GPGPU specific code. However, the steps carried out by both implementations are widely identical as can be seen by comparing both figures 29 and 30 that outline the pseudocode used to construct both the GPGPU and CPU methods.

```
Declare int for GPU position in data
Declare int for GPU stride in data

For each byte in data segment
       Declare int for state, set as initial state
       Declare int for walk, set at current position in data

       While walk is less than data segment length
              Set state according to lookup table (using state & byte of data[walk])
              If state is 0
                     Break
              If state is less than initial state
                     Record location as result and add to found results count

       Go to next byte

Synchronise GPU threads
```

Figure 29: Case study 3 GPU PFAC algorithm pseudocode

With the PFAC algorithm, processing is minimised with the use of the lookup table that acts as a state machine, which simply instructs the processor to progress searching depending on the byte read at the current position and the current state, as described in section 3.2. As the state machine drives the search, the actual processing steps are simplistic in nature when compared to the earlier algorithms adopted in this

research. The main fundamental difference of this algorithm, compared to the modified BM algorithm employed by the CPU in previous case studies, is that each and every byte of forensic data is processed in turn by an available processing thread to search for all patterns defined within the search parameters. This may sound expensive for the processor to do, however, if the first byte of the searched patterns is not discovered, the processing thread will be freed— only using a few instructions to reach that point.

```
Declare int array for found results

For each byte in data segment
	Declare int for state, set as initial state
	Declare int for walk, set at current position in data

	While walk is less than data segment length
		Set state according to lookup table (using state & byte of data[walk])
		If state is 0
			Break
		If state is less than initial state
			Record location as result and add to found results count

	Go to next byte

Return found results
```

Figure 30: Case study 3 CPU PFAC algorithm pseudocode

It is envisioned that the PFAC algorithm employed in this case study would benefit any processor that is tasked with searching for multiple patterns, as all defined patterns are searched for in a single scan of the data read from the drive. The benefit of the algorithm is imagined to significantly affect the times taken to complete in the 19 and the 40 pattern searches of the tests devised but would have less improvement on the 5 pattern searches as a multi-string searching algorithm is predicted to lose performance gain when tasked to search for fewer patterns.

Aside from the change of algorithm employed to conduct searching, this case study has no other changes to the method of searching adopted within case study 2. As this algorithm is a fundamental change to how data is searched for, it was deemed

necessary as part of this case study to keep other variables the same to obtain results which could be fairly compared to that attained in case study 2. From the performance comparison between the two studies, the difference in performance can lead to answer whether a more optimised algorithm could benefit string searching for multiple targets in the context of DF.

4.5.4 Results

Table 6 presents the results gathered from this case study. Performance improvements are witnessed across all technologies used to search for patterns within forensic data when comparing the results to that of case study 2. Indicating that all processing techniques used to process data are notably quicker with the PFAC algorithm when compared to both the brute force algorithm employed by GPGPU processing and the modified BM algorithm on CPU technologies. Comparing the times derived to the previous case study, the CPU gained the most benefit from applying the PFAC algorithm to search through forensic data, achieving a speedup across all test platforms averaging 1.15x, 3.55x, and 5.96x for the 5, 19 and 40 pattern search tests respectively. GPGPU technologies were also improved, showing speedups averaging 1.06x, 1.37x, and 1.86x, for the 5, 19 and 40 pattern tests.

What is interesting to note here is the performance growth between the 40 and 5 search pattern test from each processing technique, as the 40 search pattern tests showing the possible optimisation that a multi-string algorithm like PFAC provides when searching for larger amounts of patterns within data.

Table 6: Case study 3 search time results

5 defined patterns — Time (secs.)

Test Platform	Single CPU	Multi-CPU	Single GPU	Multi-GPU	Single IGP
A	177.82	29.21	39.64	26.91	44.81*
B	234.78	84.66	98.37	85.04	115.01
C	195.37	45.12	41.65	32.48	60.35

19 defined patterns — Time (secs.)

Test Platform	Single CPU	Multi-CPU	Single GPU	Multi-GPU	Single IGP
A	309.18	46	40.09	26.86	46.51*
B	366.1	111.05	98.78	84.97	116.28
C	343.05	83.3	42.52	32.43	63.77

40 defined patterns — Time (secs.)

Test Platform	Single CPU	Multi-CPU	Single GPU	Multi-GPU	Single IGP
A	377.34	55.05	40.44	27.06	47.34*
B	431.92	126.3	99.13	84.68	118.28
C	444.72	99.47	42.55	34.38	66.43

** - Secondary discrete GPU, no IGP present on system*

Another observation within the results collected can be observed from the improvements of the single CPU performance. All times recorded were significantly faster than single CPU results from previous studies. The single CPU results of the 19 and 40 pattern searches now outperform the base performance metrics gathered using Foremost with a speedup of 1.29x and 1.84x respectively. The 5 pattern search, however, still performs best with Foremost (0.62x)— an expected result when comparing algorithm characteristics.

One more noteworthy inspection from the times gathered is the performance of the multi-threaded CPU, which outperforms the single IGP or secondary discrete GPU in some tests when searching data with the PFAC algorithm. This can be observed from test platforms A and B in the 5 and 19 pattern searches. Within the 40 pattern searches, however, it is observed that the IGP and secondary discrete GPU retain the lead over the multi-threaded CPU as the multi-threaded CPU times are seen to depreciate with more search patterns defined.

When visualising the times gathered to analyse processing rate in figure 31, a clear overview of each processing method's performance can be seen. When comparing

the processing rate to that of case study 2, there are notable improvements in the 19 and 40 string searches, with all processing techniques deteriorating significantly less when searching for greater amounts of patterns—with discrete GPUs showing near no deterioration at all in some instances. When analysing the performance benefit on test platforms A and C, it can be seen that the 5 pattern test seems to reap less improvement than the previously used algorithms than the 19 and 40 pattern searches.

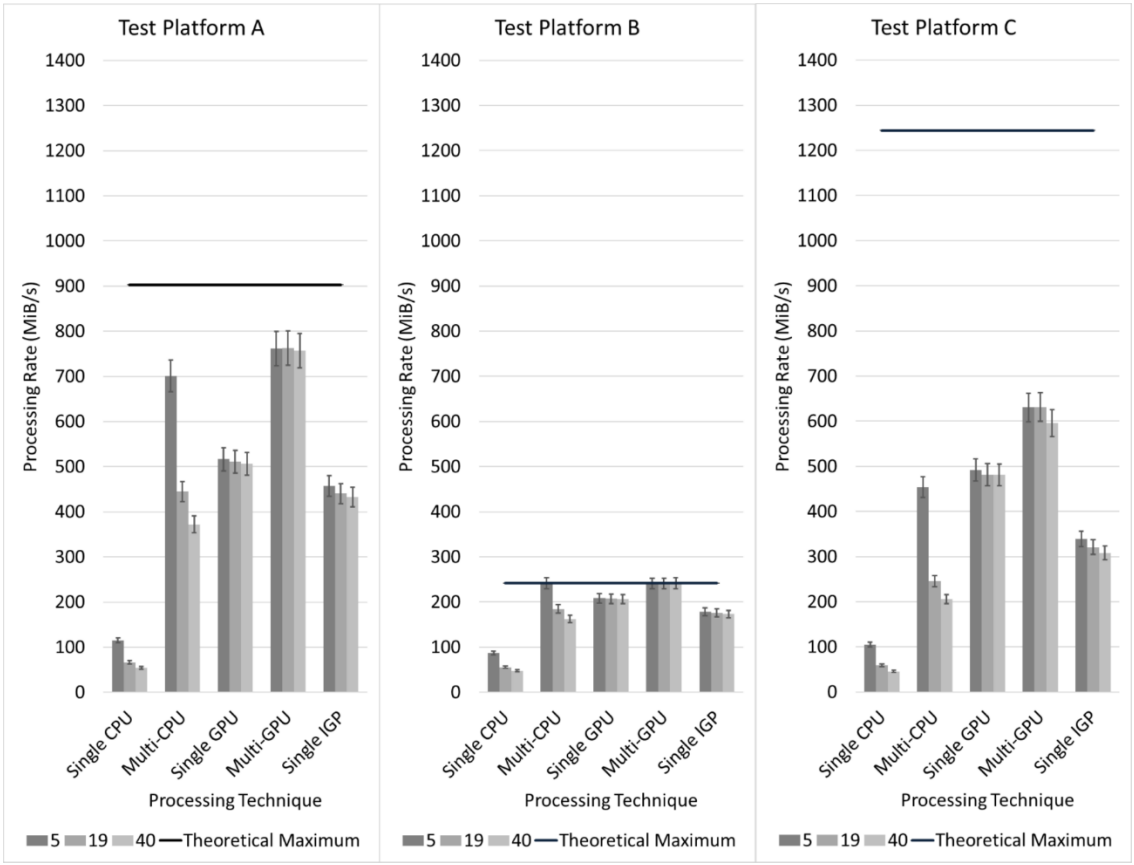


Figure 31: Case study 3 processing rate analysis with 95% confidence intervals

Processing rates derived from test platform B successfully manage to reach the theoretical maximum data transferal rate of the storage device on four counts—from all of the multi-GPU tests, and also the 5 pattern multithreaded CPU test. When analysing the performance of string searching with multiple GPUs, the multi-GPU test on test

platform B manages to employ an average 63% of the total performance of the combined performance of the individual GPGPU devices, notably lower than the average 77% of the combined performance utilised on test platform C, which similarly pairs a discrete GPU and IGP in its multi-GPU test. Test platform A, which utilises two discrete GPUs within its multi-GPU test, uses an average 80% of the combined performance of the two separate cards.

When analysing the direct comparison between the patterns searched and time is taken in figure 32, familiar patterns previously seen in the last case study reappear. While the PFAC algorithm has significantly improved the overall times to search from case study 2, the single CPU technique is seen to still require an increasing amount of time when more patterns are defined. The single CPU still shows the worst deterioration from all techniques trialled. The multi-threaded CPU showed improvements when utilising the PFAC algorithm, however, similar to its single threaded counterpart, its performance is still seen to depreciate when tasked with increasing amounts of search patterns.

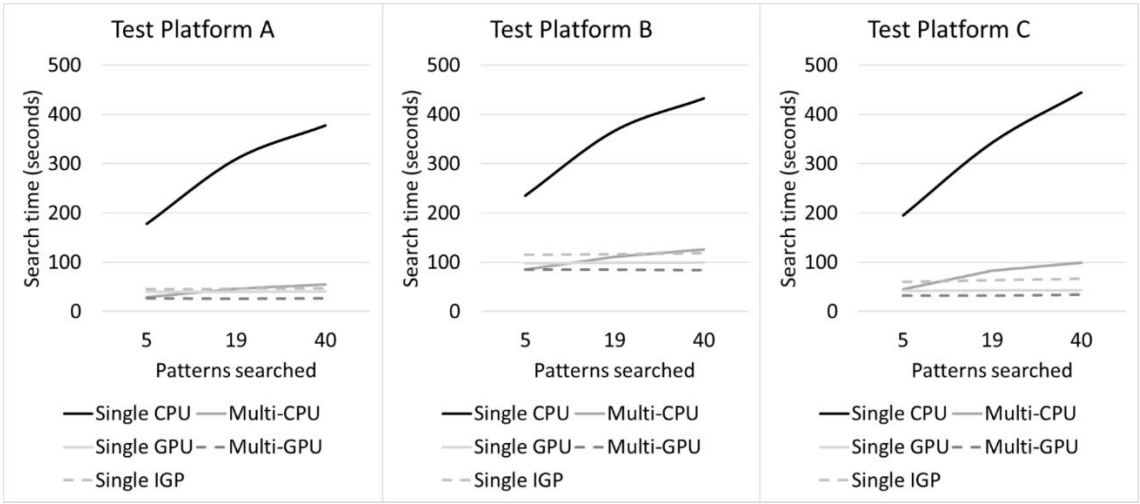


Figure 32: Case study 3 patterns searched and time analysis

The GPGPU technologies faired the best in this analysis once more. From the graphs; the multi-GPU, single GPU, and single IGP, performed all tests showing little, or no, depreciation in performance when tasked to search for increasing amount of search patterns. Evidenced in the graph, once again, is the multi-GPU technique performing the fastest of all processing techniques trialled in this case study.

When comparing the processing technique speedup over the single CPU solution in figure 33, we observe different results to that drawn by case study 2. While the multi-threaded CPU technique manages to maintain its arguably overall linear speedup over its single CPU equivalent, its speedup has notably increased with the employment of the PFAC algorithm. The multi-threaded CPU technique overall fairs better with the other GPGPU techniques on trial, managing to show larger speedups than the single GPGPU devices on the 5 pattern test, and besting the speedup of the secondary discrete GPU and IGP in the 19 pattern tests for test platforms A and B.

Test platforms A and C show the greatest variation of observed speedups, however, as mentioned earlier when analysing the processing rates of each platform, it is clear that the multi-GPU tests on test platform B are limited by storage device performance. In turn, this affects the potential speedup of the multi-GPU technique on this test platform.

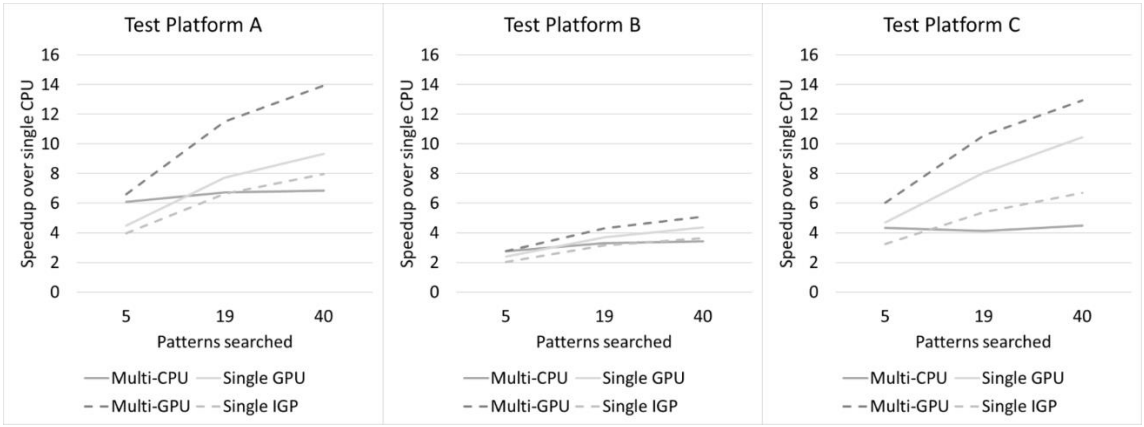


Figure 33: Case study 3 technique speedup over single CPU solution

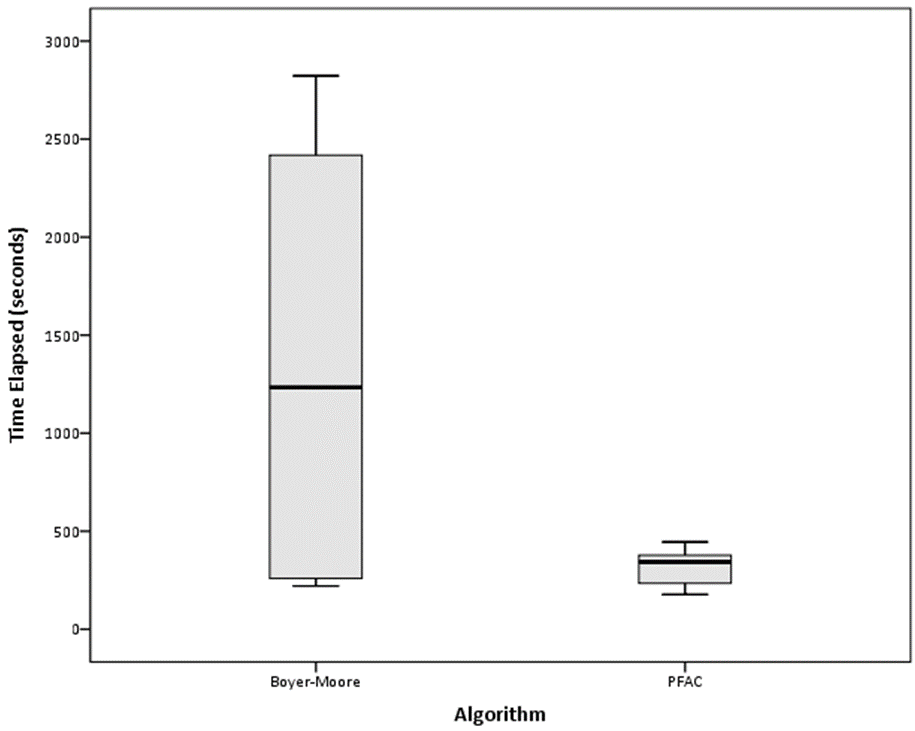


Figure 34: Average time taken for single-threaded CPU to conduct string searching with modified BM and PFAC algorithm processing

The Boyer-Moore algorithm took the longest to process data (median = 1234, min = 220 and max = 2823). The quickest processing algorithm was the PFAC algorithm (median = 343.05, min = 177.82 and max = 444.72).

A Kruskal-Wallis H test showed that there was **no** significant difference in time elapsed between the different processing techniques, $\chi 2(1) = 3.604$, $p = 0.058$.

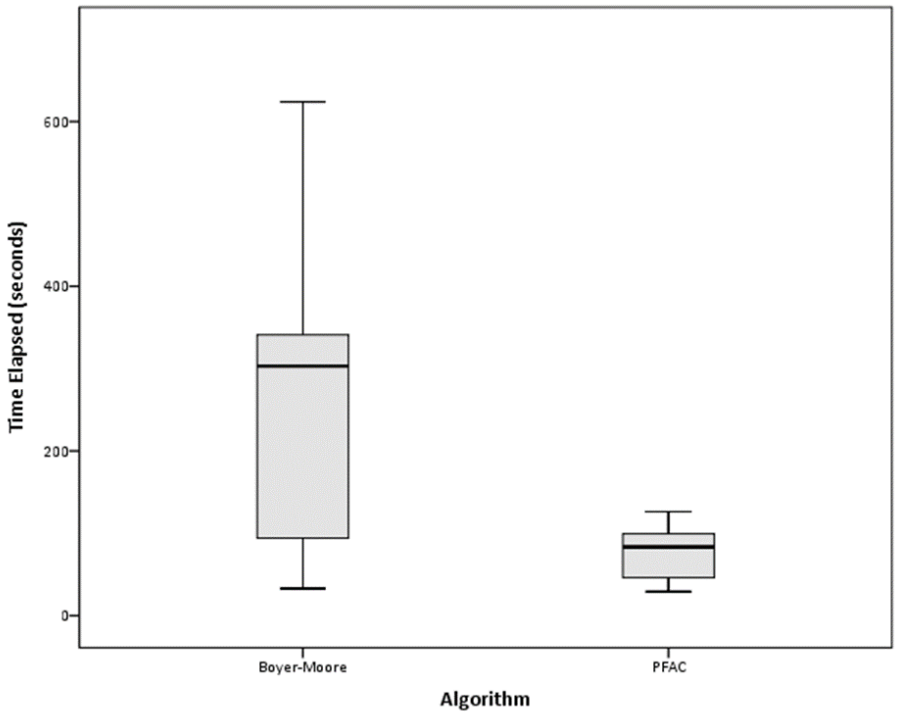


Figure 35: Average time taken for multi-threaded CPU to conduct string searching with modified BM and PFAC algorithm processing

The Boyer-Moore algorithm took the longest to process data (median = 303, min = 33 and max = 624). The quickest processing algorithm was the PFAC algorithm (median = 83.3, min = 29.21 and max = 126.30).

A Kruskal-Wallis H test showed that there was a significant difference in time elapsed between the different processing techniques, $\chi 2(1) = 4.306$, $p = 0.038$.

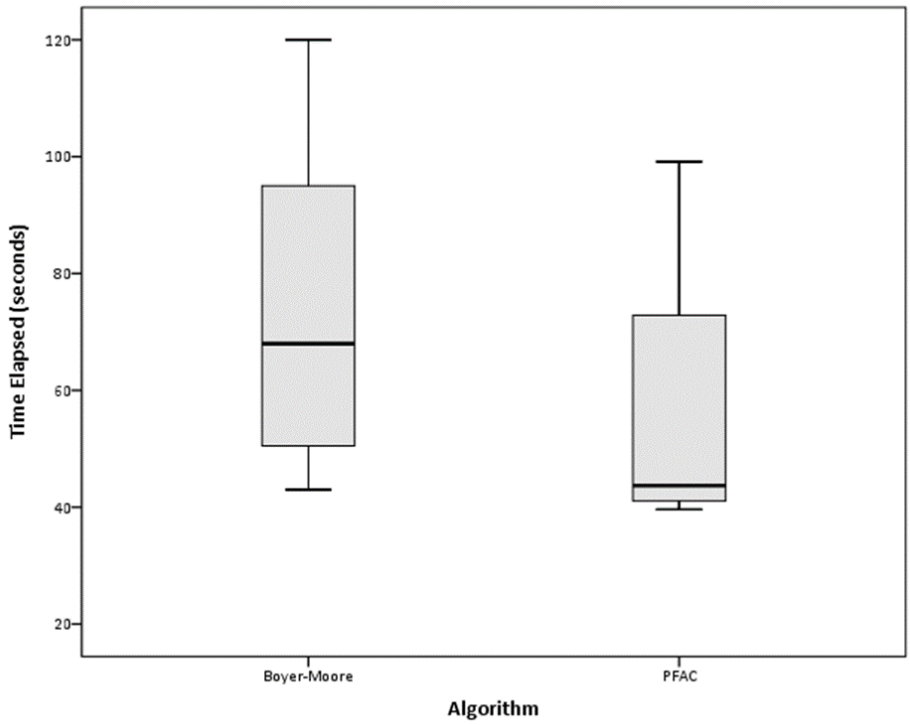


Figure 36: Average time taken for single GPU to conduct string searching with modified BM and PFAC algorithm processing

The Boyer-Moore algorithm took the longest to process data (median = 68, min = 43 and max = 120). The quickest processing algorithm was the PFAC algorithm (median = 43.68, min = 39.64 and max = 99.13).

A Kruskal-Wallis H test showed that there was a significant difference in time elapsed between the different processing techniques, $\chi 2(1) = 4.320$, $p = 0.038$.

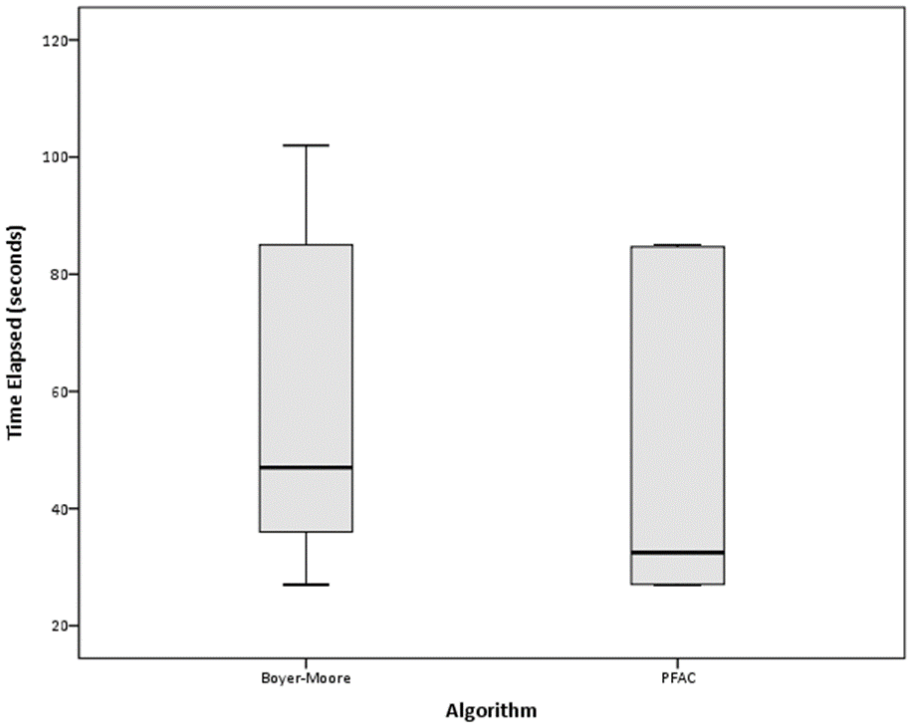


Figure 37: Average time taken for multi-GPU to conduct string searching with modified BM and PFAC algorithm processing

The Boyer-Moore algorithm took the longest to process data (median = 47, min = 27 and max = 102). The quickest processing algorithm was the PFAC algorithm (median = 32.48, min = 26.86 and max = 85.04).

A Kruskal-Wallis H test showed that there was **no** significant difference in time elapsed between the different processing techniques, $\chi 2(1) = 2.123$, $p = 0.145$.

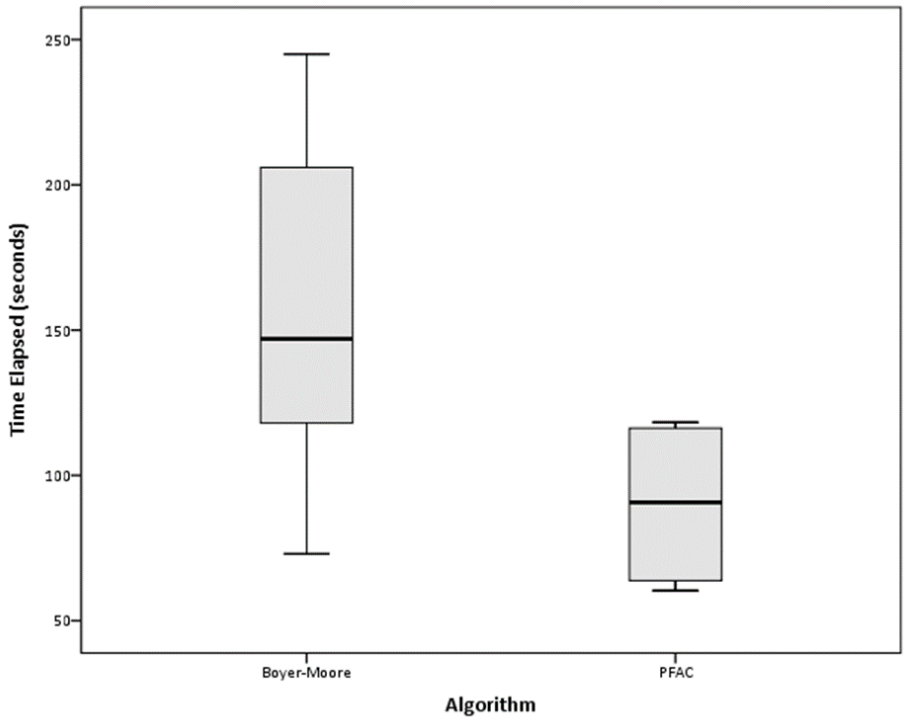


Figure 38: Average time taken for single IGP to conduct string searching with modified BM and PFAC algorithm processing

The Boyer-Moore algorithm took the longest to process data (median = 147, min = 73 and max = 245). The quickest processing algorithm was the PFAC algorithm (median = 90.72, min = 60.35 and max = 118.28).

A Kruskal-Wallis H test showed that there was a significant difference in time elapsed between the different processing techniques, $\chi 2(1) = 5.026$, $p = 0.025$.

4.5.5 Conclusions

Revisiting the predicted outcomes of this case study, it was predicted that both CPU and GPGPU searching would process forensic data faster than previous case studies when a more optimised multi-string searching algorithm was used. Overall, this prediction was correct, as statistical analysis revealed that substituting the modified BM algorithm with

the PFAC algorithm yielded significant performance improvements across most CPU and GPGPU processing techniques. The results from this case study also demonstrate relatively minor improvements over that achieved in case study 2 when searching for 5 patterns, however, more substantial improvements are attained when the algorithm was employed to search for 19, and 40 patterns.

Observations of how the PFAC algorithm affected each technology produced some fascinating results. Analysis of the times produced revealed that both the single- and multi-threaded CPU applications reaped the most benefit of the algorithm, demonstrating that the algorithm lessened the processing burden of searching for multiple strings with a more efficient state-machine driven algorithm. Likewise, GPGPU methods all benefitted from the algorithm's characteristics, allowing the processor to optimise searching of larger amounts of patterns while showing little, or no, degradation of the time required to perform searching through data.

The improvements brought by employing the PFAC algorithm also produced results which shown processing performance at the theoretical maximum transfer rate of the storage device on test platform B— largely from employing multi-GPUs to process data on this platform. Observing the single GPU and the single IGP performance of test platform B, it would be novel to presume that – if both GPGPU processors were tasked with processing forensic data from a faster storage device – the achievable processing rate observed would have been faster.

Analysing the processing rate from other platforms also pose some questions in regards to the efficiency of reading forensic data from the storage device. The storage devices on test platform A and C seemed to reduce the effectiveness of applying parallel techniques to asynchronously analyse forensic data. When testing the PFAC algorithm on all CPU and GPGPU tests, it was noted that the time that the processor took to process had become significantly quicker than the time required to load the 100 MiB segments of forensic data to the processor.

The result of this change in processing behaviour had shown that tests which employed multiple threaded asynchronous processes had many processors idling, bottlenecked by the transfer rate of the storage device serving data to other processors. However, when comparing this observed behaviour to the results produced from analysing processing rates, the witnessed delays do not coincide with the theoretical maximum performance of the storage devices tested on platforms A and C, indicating that the data transfer rate from the storage device to the processor was not reaching its fullest potential.

In order to fully investigate the delays on how forensic data was read from the storage device, additional storage device benchmarking tests were undertaken with a file reading tool developed in C#, where time was measured on how long it took to read the forensic data used within this research. This benchmarking tool utilised the same methods that OpenForensics used to read data from the storage device. Additionally, the program did not perform any further processing on the data read— the forensic data was simply read sequentially from the storage device in 100 MiB segments. After the file had been read, a time was produced displaying how long the tool took to read the data.

Benchmarking tests were performed on test platform A to confirm the suspected bottleneck, as this test platform possessed the most powerful processors of all platforms tested, and therefore more likely to suffer from the bottlenecking from reading files. Results from the benchmarking tool revealed that, by simply reading forensic data from the storage device, it achieved file reading performance of around 760 MiB per second. The measured result was lower than the measured sequential read performance measured by CrystalDiskMark and akin to the performance achieved by searching for patterns with multiple GPUs within this case study. The result recorded signifies a further requirement to investigate how forensic data is read from the storage device, to identify any other potential bottlenecks that may limit string searching performance.

4.6 Case study 4: Investigation of data reading performance

4.6.1 Introduction

Applying a multi-string algorithm improved the speed of performing string searching on forensic data significantly. Enhancements to the processing performance have raised a further requirement to investigate precisely how data is read from storage devices, as the recorded processing rate and the observed processing behaviours of case study 3 have raised uncertainties that the forensic data is not being read at an optimal rate that the storage device is capable of.

Processing behaviour witnessed during case study 3 shown that, during asynchronous threaded processing tests, freed processing threads were seen to idle—queued waiting for the storage device to pass segments to partnered threads. However, results investigating the processing rate revealed that data was not being read at the theoretical maximum transfer rate that the storage devices were capable of. The slow-down was confirmed when performing a post-analysis of case study 3, where a tool was created to simply read forensic data without performing any further processing. The test performed confirmed that the storage device was loading forensic data at a slower rate than the sequential read speed measured at the beginning of this research using CrystalDiskMark.

4.6.2 Aim

This case study aims to explore faster techniques to read data from storage devices. Through implementing a quicker method of reading from the storage device, it is anticipated that the delay observed during testing to read data will be minimised, which – in turn – will result in faster overall processing.

4.6.3 Method

Research was done that analysed how the storage device benchmarking tool used – CrystalDiskMark – measured the theoretical maximum sequential read speeds of each storage device. Research into CrystalDiskMark's operation revealed that the program serves as a front-end GUI to Microsoft DiskSpd storage performance tool, which measured the disk performance using options selected in CrystalDiskMark.

By default, CrystalDiskMark benchmarks sequential read speeds by having a queue depth of 32 on a single thread. The queue depth specifies how many dimensions of parallelism that a thread has to deploy read instructions— e.g. a queue depth of 32 would indicate that any threads tasked to read the storage device can queue a maximum of 32 read tasks at any one time. By having queued read tasks, the storage device could theoretically manage read tasks better, increasing the input/output operations per second (IOPs) and increasing performance reading and writing data.

When performing a CrystalDiskMark benchmark with the queue depth set to 1 instead of 32 – replicating the levels of parallelism employed by OpenForensics in case study 3 – test platform A records a sequential read performance of around 760 MiB per second. This signifies that, whilst synthetic benchmarks do not necessarily reflect real-world file reading performance, employing a thread without queuing read instructions may be suppressing the potential performance reading data from storage devices used in previous case studies.

To take full advantage of the storage devices within this case study, it was decided to allocate a single thread to read the data with a queue depth of 32 read instructions – akin to the default settings of CrystalDiskMark that were used to produce the benchmarks – as the settings produced data transfer rates similar to the sequential read speeds stated by the storage device manufacturer. Only a single thread was used so that it wouldn't interfere with the performance of other asynchronous threads. For multi-threaded CPU tests where all logical cores of the CPU were used to process data,

if more threads were employed to read data, it may disadvantage active processing of another processing thread. Allowing a queue depth of 32 read instructions, however, provided that single thread plentiful resources to queue enough read instructions to make full use of the IOPs of the storage devices on test.

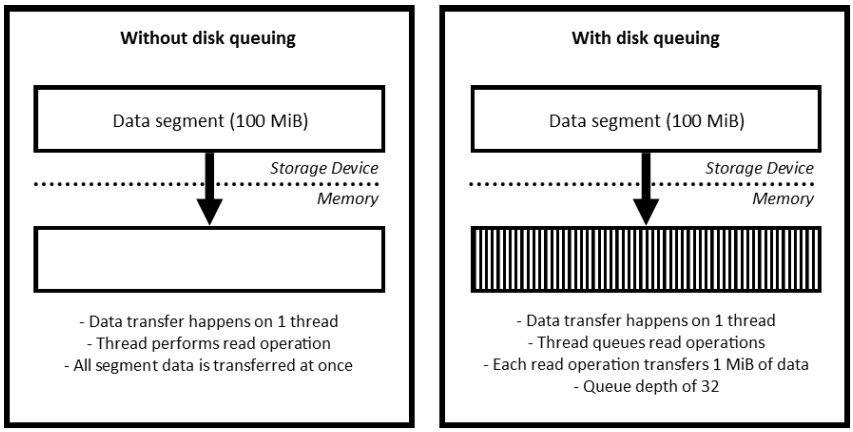


Figure 39: Data transferal differences between case study 3 and 4

As part of investigating data handling, other variables which may impend the performance of the storage device were also reviewed as part of this case study, such as; the size of the stream buffer, and the length of data read by each read instruction queued.

The stream buffer is a setting which tells the system on the rate to read data from the physical drive to memory. The stream buffer has a default rate of 4 KiB, as used within the previous case studies. This size is also set as the default used by the Microsoft DiskSpd utility. As the data from the storage device is transferred into memory by the CPU, the size of the stream buffer defined is largely limited by the size of the internal cache of the CPU. All of the CPUs tested in this research possessed fairly large internal cache, so it was assumed that by increasing the stream buffer size may have had a positive effect on data transfer speed. However, experimenting with the C# benchmarking tool on the test platforms – expanding the stream buffer to 8, 16 and 32

KiB – shown that larger stream buffer sizes did not produce faster transfer rates, as increasing the stream buffer produced no significant effects in data transfer speeds from the storage device to memory. In conclusion, it was found that maintaining a small stream buffer size of 4 or 8 KiB remains effective while maintaining compatibility with a wider range of processors. For this case study, the default rate of 4 KiB was maintained.

When reviewing the length of data read by each read instruction queued by the read thread to fill each 100 MiB segment. The sequential read tests used to measure data in CrystalDiskMark set the size of 1024 KiB for read instructions. However, an investigation conducted on how varying sizes affect the speed of data transfer on the test systems involved in this research. The C# benchmarking tool was used once again to measure the time taken to load forensic data in varying lengths. Data read by each queued task was tested in sizes of; 32, 64, 128, 256, 512, 1024, and 2048 KiB segments respectfully. Results from testing the various sizes of data read by each queued task revealed that the sizes smaller than 256 KiB performed significantly slower than the larger sizes when reading the forensic file from the storage device; additionally, larger segment sizes of 1024 and 2048 KiB segments were found to produce the most consistent results when tested on all three test platforms multiple times. From the observations of the trial, a segment size of 1024 KiB was deemed to be the optimal segment size of each queued read instruction.

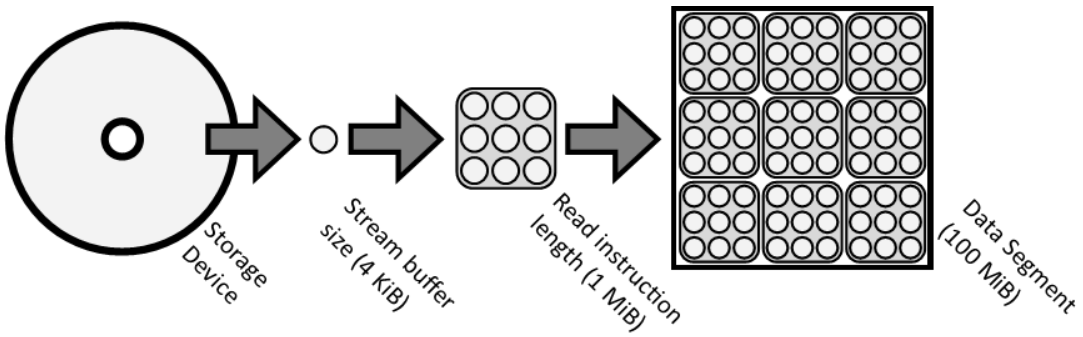


Figure 40: Case study 4 data transfer method

Figure 40 outlines the finalised design of how forensic data will be read from storage devices in this case study. The most significant difference employed by this case study is the further split of the 100 MiB data segments in 1024 KiB – or 1 MiB – sections, which are – in turn – read concurrently by queued read instructions. Illustrated by the figure are how all of the components, as previously discussed in this section, all fit together. The storage device – where forensic data is stored – is read by the CPU at a rate of 4 KiB; all of the 4 KiB 'blobs' of data, read by the CPU, fill the read instruction which is requested by the read thread; which lastly builds up the 100 MiB data segment requested by the method invoked to fetch the next segment of forensic data.

Table 7: Storage device benchmark results

Storage Device Performance (MiB/s)

Test Platform	CrystalDiskMark	C# Benchmark
A	903	974
B	242	254
C	1244	1050

Final measurements using the C# storage device benchmarking tool using the variables set produce performance results which are arguably different to that recorded by CrystalDiskMark, as shown in table 7. Despite this, as the C# storage device benchmark follows the same methods to read data as what OpenForensics employs to conduct the tests, it provides a more accurate insight of the possible data transfer speeds achievable with any processing method conducted with OpenForensics. It is reasonable, in this case, to assume that the theoretical maximum data transfer speed for each processing method trialled in this case study would be that recorded by the C# storage device benchmarking tool and not that recorded by CrystalDiskMark.

It is anticipated that the changes made to how OpenForensics reads forensic data from storage devices would reduce processing limitations imposed by utilising a

single read instruction that previous case studies used. While asynchronous search methods employed by the multi-threaded CPU and multi-GPU tests should theoretically take full advantage of the possible data transfer speeds of the storage device. It is also anticipated that the single threaded technologies will also benefit from faster data transfer speeds as less time should be required to read data. Aside from the way that forensic data is read, this case study employs no further changes to the way that OpenForensics operates within case study 3. This will allow this case study to achieve an accurate representation of how changing file reading affects the overall performance to perform string searching on forensic data.

4.6.4 Results

Results produced by applying a different file reading technique to read forensic data can be seen in table 8. In all test cases on all platforms, times were significantly improved. Comparing the differences in recorded times between this case study and results gathered by case study 3, the single CPU method of searching obtained the most benefit of the new method of reading data, as test platforms showed an average speedup of 3.1-3.5x over the single CPU results gathered by case study 3. Likewise, multi-threaded CPU tests gained a noteworthy speedup of between 1.53-2.46x over the multi-threaded CPU tests of the previous case study.

Table 8: Case study 4 search time results

5 defined patterns — Time (secs.)

Test Platform	Single CPU	Multi-CPU	Single GPU	Multi-GPU	Single IGP
A	50.27	21.17	30.21	21.07	33.37*
B	115.27	80.67	90.61	80.38	100.94
C	53.96	20.89	30.25	22.89	33.31

19 defined patterns — Time (secs.)

Test Platform	Single CPU	Multi-CPU	Single GPU	Multi-GPU	Single IGP
A	81.82	21.63	30.67	21.11	35.34*
B	144.05	81.34	94.22	80.65	108.39
C	88.82	24.49	31.46	22.92	37.88

40 defined patterns — Time (secs.)

Test Platform	Single CPU	Multi-CPU	Single GPU	Multi-GPU	Single IGP
A	101.18	22.43	30.86	21.1	35.84*
B	156.11	81.42	95.09	80.15	111.29
C	108.56	29.38	32.51	22.91	40.54

** - Secondary discrete GPU, no IGP present on system*

Comparing the results gathered by GPGPU technologies, the observed speedups achieved with the new method of searching did not reach the same levels that CPU technologies demonstrated. Nonetheless, GPGPU technologies produced results which were – on average – 1.3x better than the previous case study, which remains a noteworthy improvement over the results of the last case study.

Looking at platform B, which managed to reach the theoretical data transfer limit recorded by CrystalDiskMark in the last case study, we can see the times achieved with the multi-GPU being improved again. This signifies that further performance was gained from incorporating a different file reading method in this case study. This is further evidenced when looking at the performance analysis of the results, as shown in figure 41, which reveals that all multi-GPU tests and multi-threaded CPU tests of test platform B reached the theoretical maximum data transfer rate recorded by the C# storage device benchmarking tool.

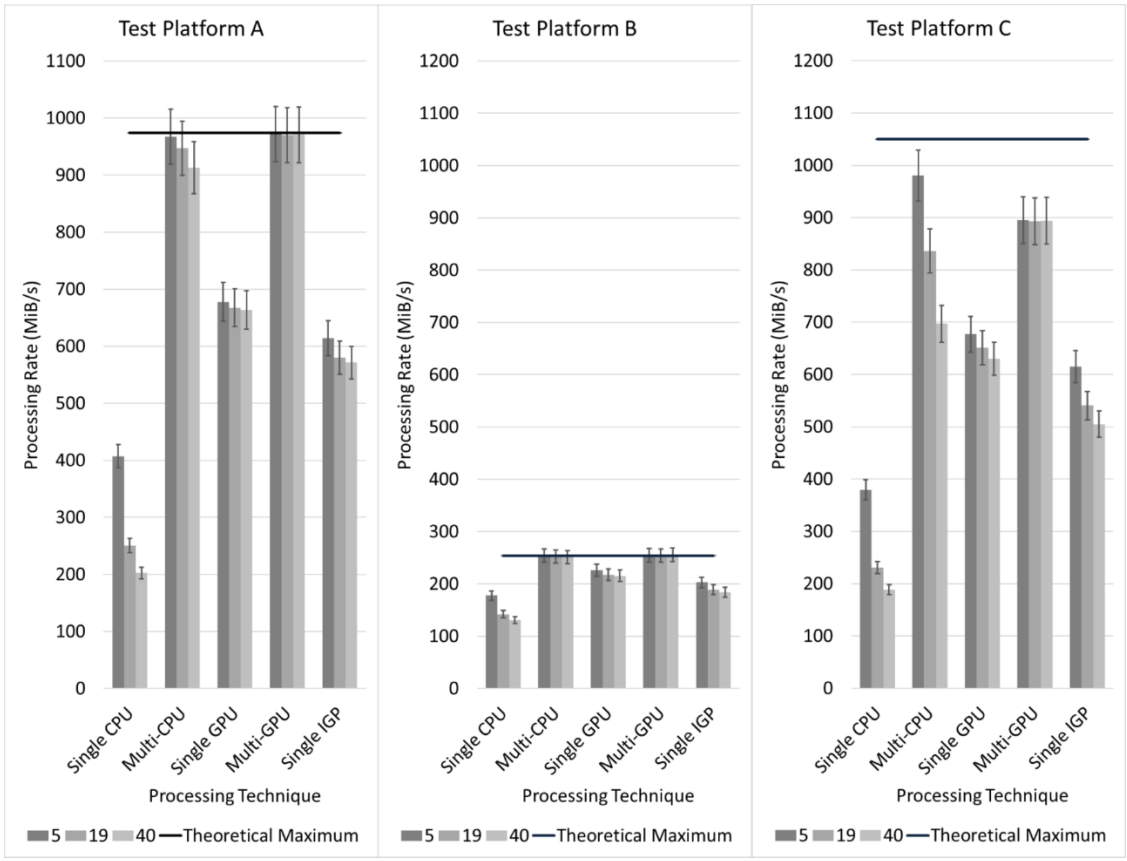


Figure 41: Case study 4 processing rate analysis with 95% confidence intervals

Further evidenced from test platform B is the growth in performance seen from the single-threaded CPU tests, showing a processing rate when searching for 5 patterns of 177.67 MiB/s, an impressive result when considering running the tests on the single IGP and single GPU achieved 202.89 MiB/s and 226.02 MiB/s respectively. However, platforms A and C found that employing a single GPU, or IGP, produced processing rates that were 200 MiB/s greater than the capabilities of the single CPU. The single-threaded processing methods performing arguably similar to each other on test platform B is likely due to the limitations of the storage device, and the synchronous nature of the single threaded tests undertaken.

Test platform A's results show that on four counts – all of the multi-GPU tests, and the 5 pattern search of the multi-threaded CPU test – seem to achieve performance

alike to the measured theoretical maximum performance of the storage device. The test platform showed that multi-threaded CPU processing performance of the 19 and 40 pattern searches deteriorated slightly, showing that the multi-GPU processing method handles searches for a greater amount of pattern better on this test platform.

From the processing rate analysis, test platform C saw the most improvement all round from the implemented changes, however, all tested processing methods fell short of the theoretical maximum processing rate which was measured by the C# benchmarking tool. The best performance of the 5 search pattern test was achieved by the multi-threaded CPU test, however, when tasked with more search patterns, the multi-threaded CPU's performance started to diminish. In comparison, the multi-GPU performance – which was the second best performing processing method on this test platform – produced similar processing performance through all pattern tests.

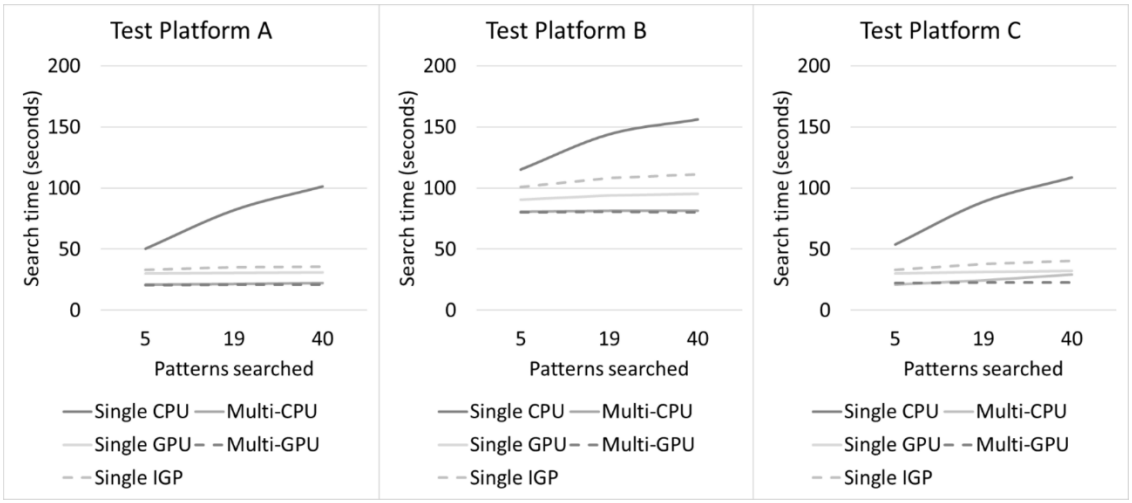


Figure 42: Case study 4 patterns searched and time analysis

Analysing the relationship between the patterns searched and time taken to search, as shown in figure 42, shows that the single CPU has produced a trend line in closer proximity to that produced by other processing methods, due to its search times

being significantly reduced with new file reading method. The single CPU still shows the most deterioration in time when tasked with increasing amounts of search patterns.

Interestingly, the new method introduced to read files has had a great impact on the time variation of the multi-threaded CPU tests, as there is less deterioration between trials than what is observed in previous case studies. The only test platform which produces any debatable growing trend line is test platform C, which is equipped with a laptop grade CPU that is susceptible to performance throttling due to thermal overheating. Nonetheless, the trend lines between multi-threaded CPU tests signifies notable improvements of the performance obtained, as the multi-CPU method is observed to perform consistently better than both the single GPU and single IGP in all trials performed.

Trends between search time and patterns searched within this case study otherwise show largely the same pattern as the previous case study, with the multi-GPU tests performing quickest in the majority of tests undertaken by all test platforms. Likewise, the multi-GPU processing method also performed the best out of all processing methods when tasked with increasing amount of defined search patterns.

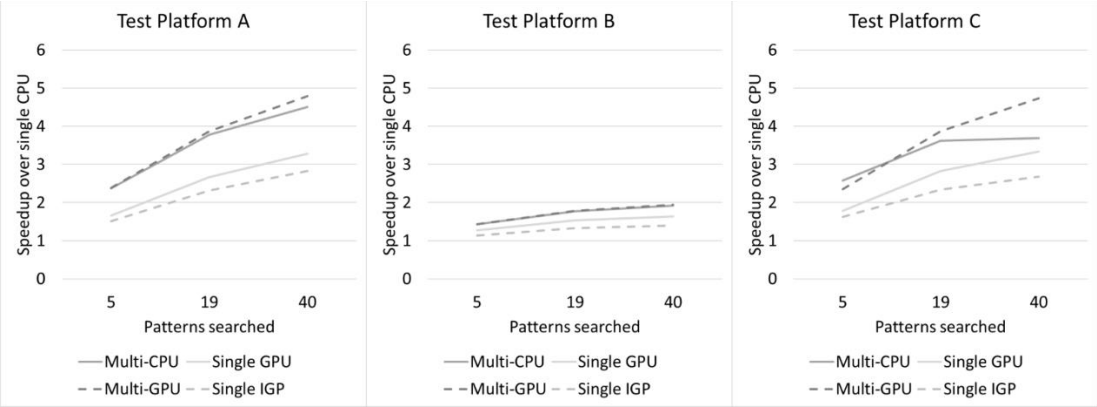


Figure 43: Case study 4 technique speedup over single CPU solution

Figure 43 illustrates the speedups observed comparing each technology's performance improvement over the results gained from the single CPU test. From this

analysis, it can be seen how significantly the performance of the multi-threaded CPU tests has been improved from the new file reading method, presenting speedups very close to that achieved by multi-GPU tests. The only oddity in the multi-threaded CPU trends is the trend line produced by test platform C, which shows the trend unusually dropping significantly within the 40 pattern search test.

Important to note within this illustration is the change of scale with the speedups, where the largest observed speedup over the single-CPU is 4.8x, which is significantly less than the maximum possible speedup observed in case study 3. This reduction in observed speedups is due to the stronger performance achieved by the single CPU in this case study.

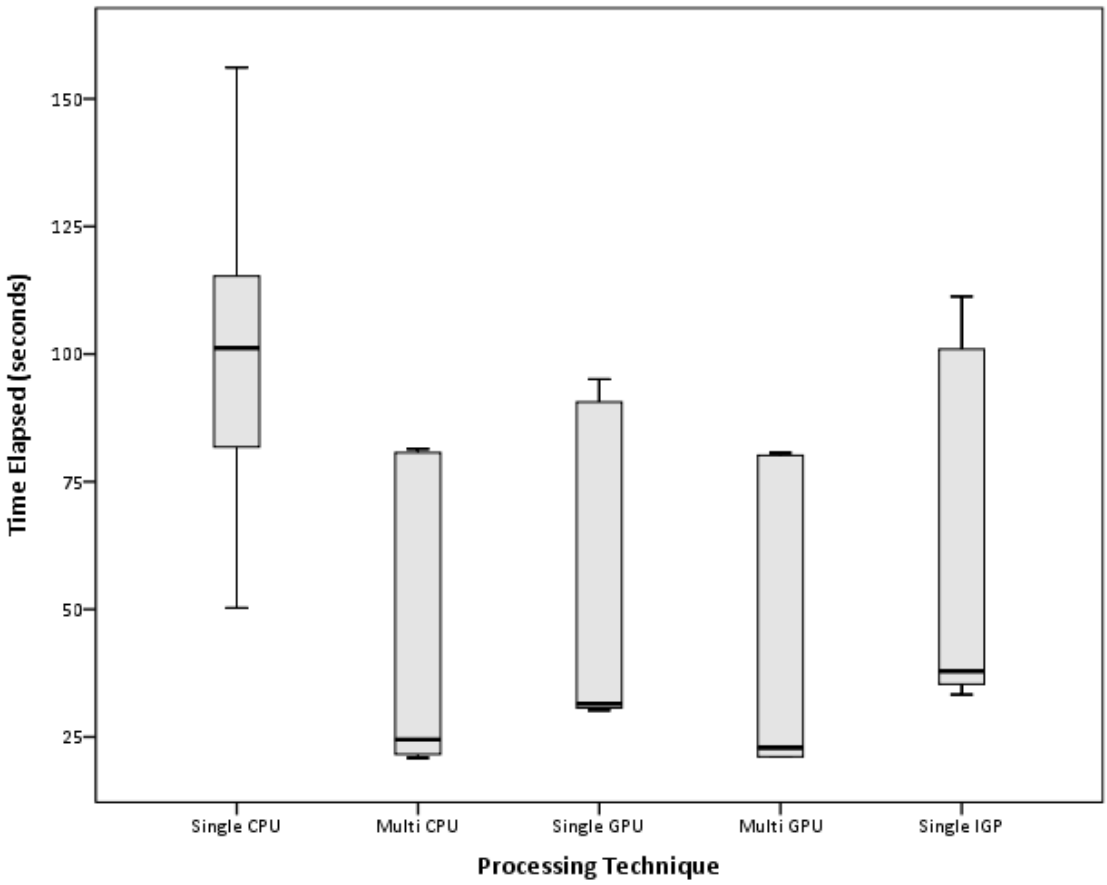


Figure 44: Average time taken to conduct string searching with each processing technique

The single-threaded CPU took the longest to process data (median = 101.18, min = 50.27 and max = 156.11), followed by the single-IGP (median = 37.88, min = 33.31 and max = 111.29), the single-GPU (median = 31.46, min = 30.21 and max = 95.09), the multi-threaded CPU (median = 24.49, min = 20.89 and max = 81.42). The quickest processing technique was the multi-GPU (median = 22.91, min = 21.07 and max = 80.65).

A Kruskal-Wallis H test showed that there was a significant difference in time elapsed between the different processing techniques, $\chi 2(4) = 19.119$, $p = 0.001$.

Follow-up tests revealed there to be a significant difference between the multi-GPU and single-threaded CPU processing techniques on time elapsed, $p = 0.002$. There was a significant difference between the multi-threaded CPU and single-threaded CPU on time elapsed, $p = 0.005$. There were no other significant comparisons.

4.6.5 Conclusions

This case study was created out of necessity after the observation of data transfer limitations within case study 3; however, this case study which investigates an alternative method of reading forensic data from storage devices has arguably produced the most unexpected results of this research. The experiments that were undertaken in this case study, which only modified the method of reading data from the storage device from case study 3, aimed to answer whether modifying how forensic data was read would have a positive effect on the time required to perform string searching. Results from all processing techniques show that the new method of reading forensic data employed in this case study achieved significantly improved performance when compared to the previous method used in case study 3.

Through employing a single thread, 32 queue depth method of reading files, the overall single- and multi-threaded CPU improvements demonstrated by this case study gained an average 2.7x speedup over the results gathered by case study 3. Likewise, single and paired GPUs took less time to search when compared to case study 3, showing an overall average speedup of 1.29x over the previous case study. This demonstrates how the technique employed to read data could significantly contribute to the overall performance obtainable when performing string searching against forensic data, as the results exemplify that the time required by the test platforms to transfer data to the processor was notably reduced.

Exploring the overall processing results further, there are no apparent relationships between the levels of parallelism between the synchronous and asynchronous tests and the level of performance speedup observed over the previous case study. However, when comparing the asynchronous multi-threaded CPU results, the speedup between this case study and the previous is seen to be quicker as fewer processing threads are observed waiting for the storage device to assign data to a partnered processor.

Overall, all processing techniques tested – in the large majority of cases – shown a greater speedup over the previous case study when tasked with more search patterns. This increased speedup when searching for larger amounts of patterns is higher on CPU tests than it is on GPGPU tests. This finding presents evidence suggests that, although all processing techniques have benefitted from the improved method to read data, the time to conduct searching is still largely influenced by the processor's processing power.

The theoretical maximum data transfer speed for this case study was recorded by using the C# storage device benchmark tool, which recorded how long it took to read data using the same methods employed in OpenForensics. The maximum data transfer speed was achieved through performing string searching on multi-threaded CPU, and multi-GPU, techniques on the two desktop computers tested— test platforms A and B. The multi-GPU on both platforms showed no deterioration in search time when tasked with greater amounts of patterns; however, performing searches for increasing amounts of search patterns using a multi-threaded CPU on test platform A did show performance waning. This indicates that whilst the multi-threaded CPU method could effectively handle lower amounts of search patterns, performing searches for greater amounts of strings may still more efficiently processed with a multi-GPU method.

Through applying a new file reading method using a concurrent queue system for data reads, OpenForensics was also able to perform string searching faster on two test platforms than the theoretical maximum storage device transfer speeds recorded by CrystalDiskMark. This was unforeseen as the parameters used to read data from the

storage device are largely comparable to the parameters used by CrystalDiskMark to perform benchmarking.

There could be many factors behind this observation. However, it is acknowledged that the variances in performance may potentially be due to differences in the underlying methods that OpenForensics and Microsoft DiskSpd – used by CrystalDiskMark – use to handle data transfers. Microsoft DiskSpd is a tool developed in C++ and possesses no .NET platform dependency to perform file reading, as all read operations are performed by methods defined within the tool's source code using direct IO to access the storage device. OpenForensics, however, employs .NET 4.0's FileStream class to perform file reading operations and benefits from using a buffer to read data from the drive.

The single CPU performance was also seen to surpass the performance measured using Foremost within this case study, whereas before, the single CPU performance was observed to be notably worse— even within case study 3, where a better multi-string searching algorithm was employed to conduct searching. This would suggest that performance was being restricted in all previous case studies by the method used to transfer data from the storage device into memory. It is reasonable to assume that, if the same file reading technique were to be applied to earlier studies, similar performance gains observed between this case study and case study 3 may also be gained.

Applying the file reading technique to earlier case studies remains a frivolous task, as performance between algorithms – shown in case studies 2 and 3 – demonstrated that the PFAC algorithm was better suited than both the modified BM algorithm and a brute force GPGPU algorithm in performing string searching. These findings remain relevant, and conducting further tests with the improved file reading method would likely demonstrate the same findings.

While it was anticipated to see performance enhancements across all processing methods trialled, the substantial improvements produced over the previous case study

suggest that processing power and efficient algorithms are not sole factors in achieving the best performance out of a storage device. This case study produces results which suggest that the technique employed to read files from the storage device remains equally important as the other factors mentioned.

Table 9: Case study 4 speedup over base performance metrics gathered by Foremost

5 defined patterns — Speedup over Foremost

Test Platform	Single CPU	Multi-CPU	Single GPU	Multi-GPU	Single IGP
A	2.27	5.38	3.77	5.41	3.42*
B	1.39	1.98	1.77	1.99	1.59
C	1.93	4.98	3.44	4.54	3.12

19 defined patterns — Speedup over Foremost

Test Platform	Single CPU	Multi-CPU	Single GPU	Multi-GPU	Single IGP
A	5.07	19.19	13.53	19.66	11.74*
B	3.14	5.57	4.81	5.62	4.18
C	4.95	17.97	13.99	19.20	11.62

40 defined patterns — Speedup over Foremost

Test Platform	Single CPU	Multi-CPU	Single GPU	Multi-GPU	Single IGP
A	7.32	33.04	24.01	35.12	20.68*
B	4.87	9.35	8.00	9.49	6.84
C	7.30	26.96	24.36	34.57	19.54

** - Secondary discrete GPU, no IGP present on system*

As it could be argued that performing string searching against forensic data is not entirely grounded in the academic field of DF, but rather computer science in general. This series of case studies presented in this research has so far aimed to dissect, improve and measure string searching performance in a bid to improve upon the overall processing rate in which DF tasks could be performed. To this end, this has been accomplished, as comparing string searching times gathered from this case study to the base performance metrics gathered using Foremost show performance speedups as presented in table 9. Statistical analysis also shows that employing asynchronous multi-CPU or multi-GPU techniques are significantly faster than employing a single CPU to

conduct string searching. Within the next case study, this research aims to present how the proposed improvements to how string searching is conducted affect the overall performance of performing file carving.

4.7 Case study 5: Applying proposed string searching methods to conduct file carving

4.7.1 Introduction

This research presents significant improvements to how string searching is conducted that also caters to the scientific accuracy necessitated by DF investigation. The changes presented are the outcome of investigating; the process which drives searching through forensic evidence, the application of a multi-string searching algorithm to conduct searching, and the method which forensic data is transferred from the storage device to memory. The results produced from applying revisions in the areas above have shown substantial growth in performance between each case study. When conducting string searching, the latter of the case studies – case study 4 – demonstrates performance speedups of up to 35.12x when compared to results gathered using an existing DF file carving tool— Foremost.

4.7.2 Aim

This final case study aims to investigate how improvements to string searching will impact the time required to perform file carving on the forensic data used in previous case studies. The processing framework used in case study 4 will be used to accelerate the string searching operations.

4.7.3 Method

In this test, file headers and footers will both be searched, then found files will be reconstructed from the forensic data and saved back to the storage device used to read forensic data. This case study varies from past case studies by introducing the

requirement to search for matching footers for each header searched for. Due to this, it requires this case study to undertake a different set of patterns to search for.

Table 10: Case study 5 patterns searched

File Type	File Header (bytes)	File Footer (bytes)
jpg	FF D8 FF E0 00 10	FF D9
jpg	FF D8 FF E1 35 FE	FF D9
gif	47 49 46 38 39 61	00 3B
gif	47 49 46 38 37 61	00 3B
png	89 50 4E 47 0D 0A 1A 0A	49 45 4E 44 AE 42 60 82
mpg	00 00 01 BA	00 00 01 B7
mpg	00 00 01 B3	00 00 01 B7
docx	50 4B 03 04 14 00 06 00	50 4B 05 06
pdf	25 50 44 46	0A 25 25 45 4F 46

The tests conducted in this case study will search for and reconstruct 9 file types. A file type is defined by this case study as a file which possesses a unique file header. All the file types searched for are presented in table 10. As some of the file types searched for share the same file footer – which indicates an end of a file – the overall amount of patterns searched for within this case study is 15. Duplicate strings do not require to be searched for individually.

Alongside the change in the defined search patterns, this case study modifies the processing cycle utilised by OpenForensics, introducing a check to see if there are any found patterns within the file segment before retrieving the next segment of data from the storage device. The revised processing cycle used for OpenForensics in this case study is presented in figure 45. If there are any found patterns in the processed segment, that segment is handed to a CPU thread which extracts all files within the data segment by using the array of file headers and footers found. Once the extraction thread finishes, the processing thread will request another segment of data. The maximum possible file size set for all searched file types was set to 10 MiB, indicating that if a footer were not

found in the first 10 MiB after the header position, the program would extract the 10 MiB of data from the header location and label the file as incomplete.

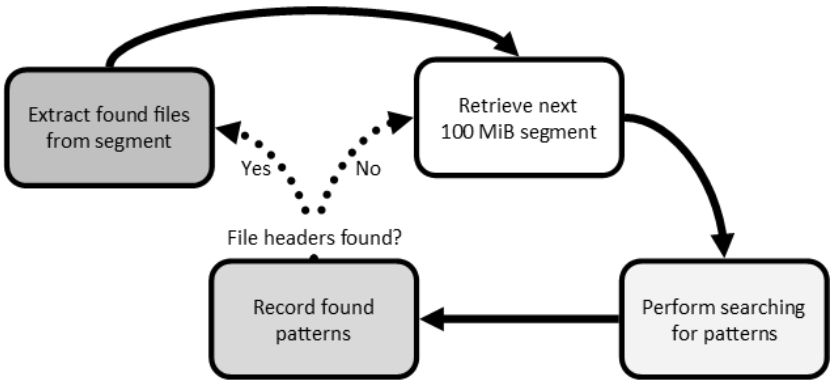


Figure 45: Case study 5 processing cycle

For Foremost, testing was done by using the command stated in figure 46, which is identical to the previous command used to gather base performance metrics in Foremost, however, omits the use of the *-w* flag. This instructs Foremost to reconstruct all found files within forensic data analysed and produce an audit file outlining the results of the file carving. The configuration file used – documented in appendix B.4 – specifies all file types and headers outlined within table 10, and specified a maximum file length of 10 MiB for all file types defined, identical to that set by OpenForensics. Foremost behaves in the same fashion as OpenForensics when a matching footer is not found for the header in forensic data. For this case study, Foremost is set to extract 10 MiB of data from the found header and marking it as an incomplete file if a matching footer is not found.

```
foremost -i TestImage.dd -c /cdrom/foremost/foremost.conf -o ./foremost
```

Figure 46: Case study 5 Foremost command

Results from testing file carving on forensic data are predicted to produce similar findings from the previous string searching tests of case study 4, as the possible processing performance in conducting file carving is highly dependent on the ability to efficiently perform string searching. Nonetheless, it is noted that the time required to extract found files will give processing techniques with higher degrees of asynchronous parallelism an advantage in this test, as it is envisioned that file reconstruction will potentially stall threads from processing other segments of data. By employing more processing threads, it is theorised that the stall in extracting files will be less noticeable due to other processing threads being able to occupy data transfers from the storage device more efficiently.

4.7.4 Results

Times recorded to perform file carving for the 9 defined file types are presented in table 11. Expectedly, the time required to perform file carving – which performs more processing operations than string searching – took between a 5 and 19 pattern string search in Foremost testing. However, contrasting performance was observed from all processing techniques tested on the OpenForensics software platform, whereas the time required to complete file carving was observed to be slower than the performance of a 40 pattern string search.

Table 11: Case study 5 file carving time results

9 File Types (15 patterns) — Time (secs.)

Test Platform	Foremost	Single CPU	Multi-CPU	Single GPU	Multi-GPU	Single IGP
A	244	145.53	28.5	51.6	35.77	61.17*
B	282	206.41	83.02	117.27	87.38	146.59
C	217	154.87	40.51	59.26	44.2	74.29

** - Secondary discrete GPU, no IGP present on system*

Regardless of the performance differences between performing string searching and file carving on the respective platforms, it was found that the time required for all processing techniques trialled on OpenForensics were significantly faster than the time taken with Foremost. Analysing the different processing techniques tested with the OpenForensics platform, it was also found that the multi-threaded CPU technique performed the best on all three test platforms, outperforming the result of the asynchronous multi-GPU method.

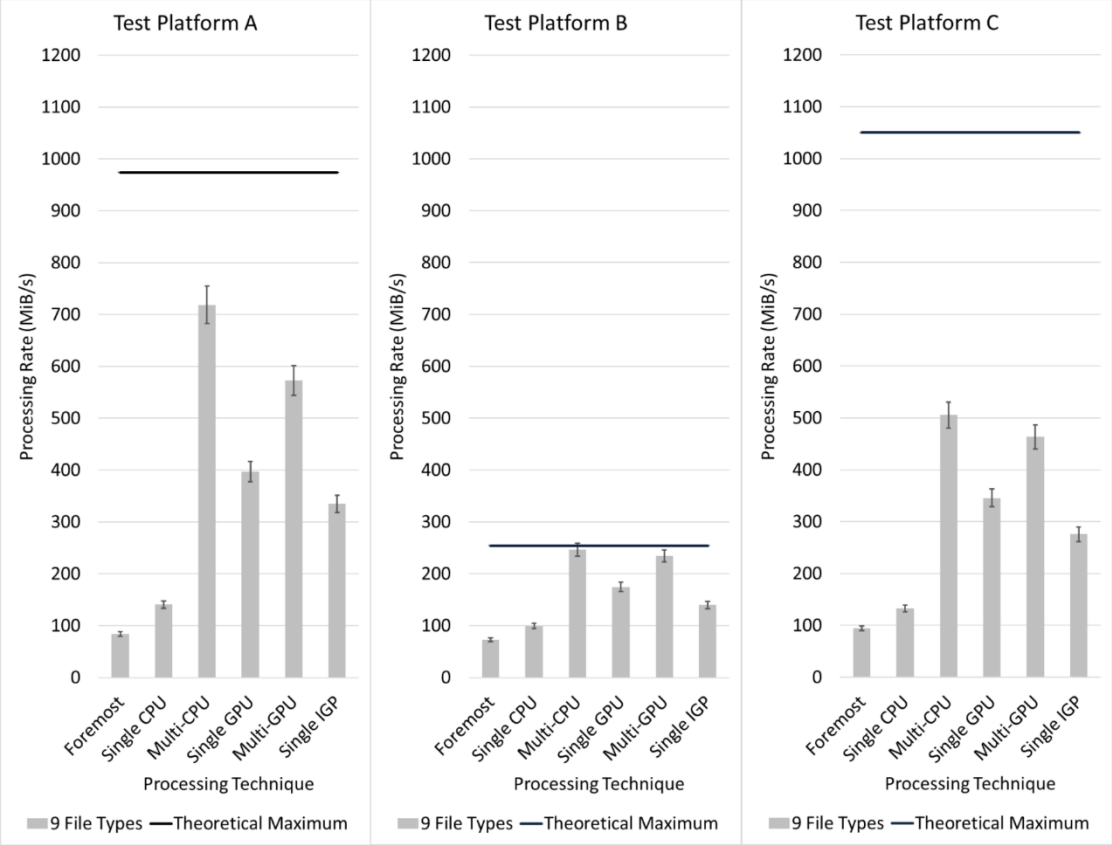


Figure 47: Case study 5 processing rate analysis with 95% confidence intervals

When performing processing rate analysis in figure 47, we get a clearer picture of how the times took affect the performance of each platform. It is found that the performance delivered by Foremost is the worst on all test platforms, performing

between 72-94 MiB/s. OpenForensics results for the single-threaded CPU test achieved between 99-140 MiB/s in comparison. When comparing the best performing techniques, performance varies between 246-719 and 234-573 MiB/s for the multi-threaded CPU and multi-GPU respectively.

Test platform B came closest to achieving the theoretical maximum performance of the storage devices on test when performing file carving. However, other platforms which possessed much faster-performing storage devices did not.

Table 12 presents how much speedup over Foremost that each processing technique utilised by OpenForensics achieved. The greatest speedups observed were attained by the multi-threaded CPU tests, which showed speedups over Foremost by up to 8.56x. Multi-GPU tests likewise shown notable improvements over Foremost's file carving performance, demonstrating speedups of up to 6.82x.

Table 12: Case study 5 speedup over Foremost results

9 File Types (15 patterns) — Speedup over Foremost

Test Platform	Single CPU	Multi-CPU	Single GPU	Multi-GPU	Single IGP
A	1.68	8.56	4.73	6.82	3.99
B	1.37	3.40	2.40	3.23	1.92
C	1.40	5.36	3.66	4.91	2.92

** - Secondary discrete GPU, no IGP present on system*

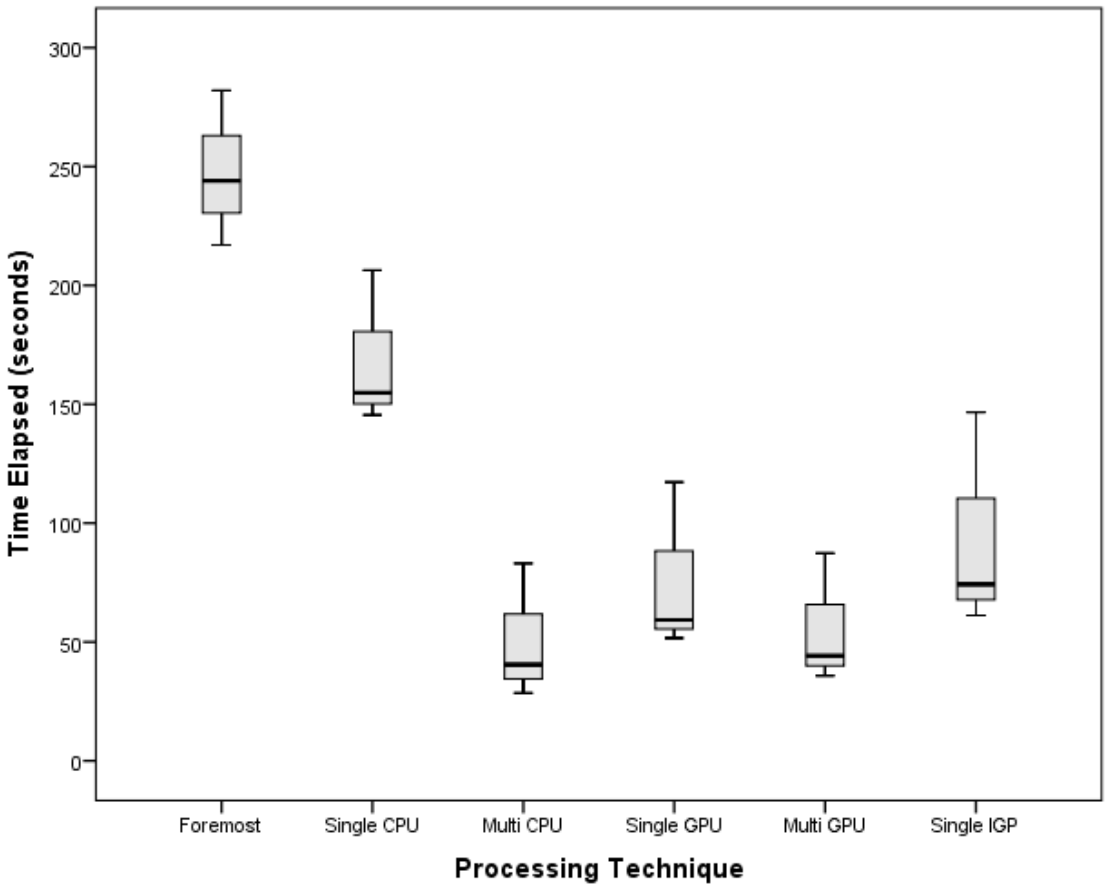


Figure 48: Average time taken to perform file carving with each processing technique

Foremost took the longest to process data (median = 244, min = 217 and max = 282), followed by the single-threaded CPU (median = 154.87, min = 145.53 and max = 206.41), the single-IGP (median = 74.29, min = 61.17 and max = 146.59), the single-GPU (median = 59.26, min = 51.6 and max = 65.67), the multi-GPU (median = 44.2, min = 35.77 and max = 87.38). The quickest processing technique was the multi-threaded CPU (median = 40.51, min = 28.5 and max = 83.02).

A Kruskal-Wallis H test showed that there was a significant difference in time elapsed between the different processing techniques, $\chi 2(5) = 12.883$, $p = 0.024$.

4.7.5 Conclusions

The aim of this case study was to establish how much of an affect the improvements to string searching would have on the performance of completing file carving against forensic data. Results from this case study have shown significant improvements when file carving 9 different file types. The largest of the speedups were attained from the multi-threaded CPU method of file carving, which demonstrated up to 8.52x speedups over the results derived from Foremost. All of the processing techniques tested with OpenForensics managed to surpass the performance of Foremost, indicating that better performance can be achieved by applying a combination of; asynchronous parallelism, multi-string searching algorithms, and an enhanced file reading technique.

The file carve test involved in this study searched for 15 unique patterns, placing this test between the 5 and 19 string tests of the previous string searching case studies. While it is true that the multi-threaded CPU performance exhibited the best performance conducting file carving in this test, results from case study 4 demonstrate that the performance of a multi-threaded CPU solution would deteriorate when asked to carve greater amounts of files. Therefore, it is determined that, although performing slower than multi-threaded CPU in this test, multi-GPUs will have the greatest benefit within performing processor intensive DF operations with greater amounts of file types.

It was observed during testing with OpenForensics that the levels of asynchronous parallelism had a positive effect on the ability to better use the data transfer performance of the storage device. The introduction of recreating files from data segments presented further delay for the processing thread as it waited for the file reconstruction operation to finish before moving to analyse a new segment of data. When more processing threads were applied to perform file carving, the effect of this delay was less noticeable, as there were sufficient processors available to queue to read file segments from the storage device. The multi-GPU tests of the test platforms, which only employed two asynchronous processing threads, were at a disadvantage when

compared to the multi-threaded CPU tests, which employed one asynchronous thread for each of the 4 to 12 logical CPU cores.

It is recognised that if more asynchronous threads were created for the multi-GPU method, whether through the employment of more GPU hardware or allowing the GPU to process more than one segment at a time, further performance could have been achieved. This has been identified by the author for future work within the area.

While it may also be true that the delay of reconstructing files from the current data segment could have been mitigated by allowing the processor to load another segment into a new location of memory, it may have introduced the danger of causing the memory stack to overflow due to the stockpiling of segments requiring file carving. It is predicted that better method of performing file carving would have been to produce a map of where all of the discovered files were, then conduct a second pass through the data to reconstruct files. Testing another method of file carving, however, was deemed to be outside of the scope of this research. It is envisioned that through utilising this approach, it may have produced results closer to that demonstrated within the string searching case studies.

In closing remarks, this case study successfully showed that applying an asynchronous parallelised model using a PFAC algorithm and enhanced file reading technique improved performance over Foremost when tasked to perform file carving of 9 different file types. It is envisioned that, within future work, investigating better file carving strategies may further enhance the performance seen through applying these methods.

4.8 Case study conclusions

4.8.1 Summary of case study results

To answer the research questions presented, a software platform – OpenForensics – was created to conduct string searching and file carving similarly to a currently available open-source file carving tool— Foremost. Four case studies were designed and presented showing the effects of introducing varying changes to the methods that Foremost utilised to perform string searching of forensic data, with the fifth case study measuring and presenting how these improved string searching methods affected file carving performance.

The first case study, which introduced GPGPU processing with a basic brute-force algorithm, showed significant improvements that exceeded the single threaded CPU performance presented by OpenForensics as well as Foremost. This remained true even when utilising the less powerful IGP processor onboard the CPU. Changes were also made to the program's processing operation, which introduced a reactive rather than proactive approach for checking for pattern matches which may be split between data segments. However, this was later deemed to be an inefficient approach when introducing asynchronous parallelization, as the proactive approach may have caused delays searching for historic data on traditional HDDs. Lastly, this case study introduced another change by increasing the data segment size, but it was later found that there were no significant benefits observed when comparing comparative results from the single threaded processing techniques to that achieved with case study 2. Like the deduction of the reactive processing approach, the increase of segment sizes may have posed memory issues or delays reading data when massively parallelised.

Comparable to the performance gained from the first case study, the second case study demonstrated significant improvements in processing performance once more through the incorporation of asynchronous threaded processing methods. This

case study also reverted to using the same segment size and processing method used by Foremost. While the results of the single threaded processing techniques were largely like the comparative results of case study 1, the second case study showed noticeable improvements when running both CPU and GPU technologies asynchronously, employing all available logical processors as separate independent processing threads. When employing an asynchronous multi-GPU approach with 5 search patterns, slower storage devices on test were seen to limit the achievable performance of the device, providing early evidence that an insurmountable bottleneck may limit the achievable processing rate.

The third case study shows the effect of applying an optimised multi-string algorithm, PFAC, to conduct string searching on digital evidence. While in most instances the general performance was improved across all test platforms, the most noticeable improvements could be seen when searching for larger amounts of search patterns. When searching for more patterns, the performance searching with the PFAC algorithm deteriorated significantly less than the CPU's modified BM algorithm and the GPGPU's brute force algorithm used in the previous study. The asynchronous multi-threaded CPU tests of this case study were seen to beat that of a single synchronous GPU when searching for 5 patterns. However, the single GPU was quicker than the multi-threaded CPU when tasked to search for 19 and 40 search patterns. The multi-GPU on the slower storage device was observed to fully utilise the storage device performance on all pattern tests, strongly suggesting that the sequential read speed of the storage device is the maximum processing rate achievable by any processing technique.

The results of case study 3 showed inconsistencies between the recorded results and the observed behaviour during testing. The storage device during multi-threaded asynchronous tests was seen to be constantly transferring data on test platforms A and B, however, the achieved performance from the asynchronous tests on these platforms did not reflect upon the observed behaviour, indicating that the data was not being transferred efficiently from these storage devices.

Case study 4 aimed to address these concerns by introducing a different method to load data from storage devices. The method used in this case study reflected and applied techniques used by benchmarking software to read data from the storage device. The resulting times derived from testing showed significant improvements across all test platforms on all tests. This was due to the storage device being able to read data at a faster rate than before, causing processors to idle less than previously seen in case study 3. All processing rates achieved, once again, did not exceed the theoretical maximum transfer speed of the storage device, however, on some occurrences, both the multi-threaded CPU and multi-GPU met the theoretical maximum processing rate when conducting string searching.

The final case study aimed to measure how the proposed changes to conducting string searching affected the speed of performing file carving of forensic data. For this case study, the techniques used in case study 4 were benchmarked against Foremost to perform file carving, where found files within the forensic data were reconstructed and saved back to the storage device. It was found that the processing techniques produced in case study 4 performed significantly better than Foremost when performing file carving 9 file types consisting of 15 unique patterns.

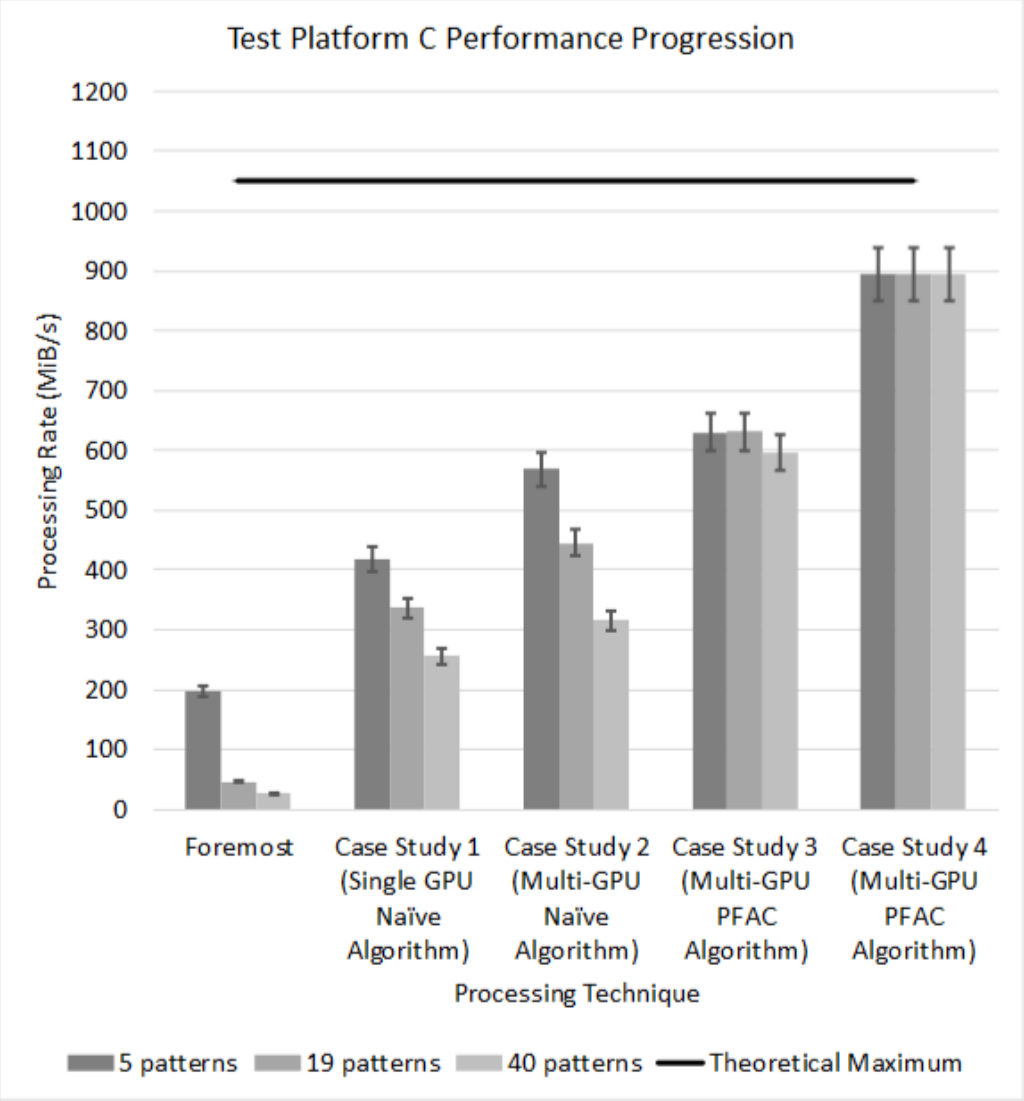


Figure 49: Test Platform C performance progression

4.8.2 Validation of research in a real-world digital forensics scenario

Testing conducted with a prototype tool based on the solution of case study 2 was trialled by the digital forensics division of Police Scotland in November 2015. The test carried out by Police Scotland involved analysis of a 120GB storage device connected to a workstation PC. The storage device was analysed by software used by the forensics division as well as a prototype file carving version of OpenForensics. Both the software

used by Police Scotland and OpenForensics were reported to have the same search criteria used to perform file carving.

Feedback received from Police Scotland stated that the OpenForensics tool was, on average, 160% faster than an equivalent product used by the forensics division. Feedback also further commenting that the equivalent product used for comparison only selectively searched around 50% of the forensic data on the drive, whereas OpenForensics searched the full volume of forensic data (appendix C).

Unfortunately, after reaching out for more information, Police Scotland was not very forthcoming with any other information regarding the tests performed or the equivalent tool used. Regardless, the feedback received by Police Scotland validates the methods and approach proposed by this research surpass equivalent tools used by Police Scotland for performing file carving. It is envisioned that the developments and improvements adopted and evidenced by later case studies would significantly improve upon the reported performance.

At the time of writing this thesis, no further trials have been conducted with Police Scotland, however, it is planned that a further prototype tool would be distributed freely to Police Scotland and other digital forensic institutions for further trials. It is also envisioned that further in-depth case studies will be completed to compare the developed OpenForensics processing framework with commercially available digital forensic tools.

Chapter 5: CONCLUSION AND FUTURE WORK

5.1 Conclusion

This conclusion will revisit the research aim of whether the application of GPGPU technologies and modern parallelisable string searching algorithms could reduce the time required to perform file carving in DF investigations. This will be answered by addressing the research questions on; whether the OpenCL GPGPU framework was reliable and quick in analysing forensic evidence, if there were advantages of employing GPGPU processing over CPU processing methods, and whether there were any benefits of applying multiple GPGPU devices to perform pattern matching. The conclusion will also answer whether modern parallelisable algorithms are better suited for the requirements of DF investigation, and whether the potential performance file carving is limited by storage device performance.

5.1.1 Research question Q1

This research has attempted to answer whether OpenCL GPGPU framework provides a reliable foundation to analyse digital evidence and decrease the time required for processing forensic images without affecting accuracy. From the evidence presented in all case studies (1-5), this research can confidently claim that it does, further adding that it is well-suited to the exploratory nature of DF investigation due to its ability to search for large amounts of patterns with less time detrition than multi-threaded CPU options.

Throughout all case studies (1-5), utilising OpenCL and GPGPU devices shown perfect reliability and accuracy throughout all tests, returning results identical to OpenForensics CPU driven techniques, and matching the results derived from Foremost. Therefore, the research presents that there are no disadvantages to reliability and accuracy when employing OpenCL in DF tools.

5.1.2 Research question Q2

The decision of choosing the OpenCL programming language to operate GPGPU devices was to increase compatibility beyond only one device vendor, albeit, at a small cost to performance. This was done so that all available GPGPU devices available on that system would be able to aid the processing of digital evidence— making full use of the computational power available. The ability to share processing workload across the available GPGPU devices on a system has proven very beneficial when adopted in case studies 2-5, showing substantial improvements over employing a single GPGPU device.

In case study 4, asynchronous multi-GPU processing was found to fully utilise the available theoretical processing rate from test platforms A and B, whilst also achieving ~85% utilisation on test platform C. Compared to singular GPGPU operation, multi-GPU string searching was seen to similarly depreciate significantly less than CPU counterparts when searching for larger amounts of patterns. Whilst it is acknowledged within this research that further research into the asynchronicity of the GPGPU approaches would benefit performance, multi-GPU processing would arguably allow for more processing capacity—future proofing pattern matching techniques within DF tools.

5.1.3 Research question Q3

The results from all case studies (1-5) show that utilising GPGPU devices show less time depreciation when searching for larger amounts of search patterns than using CPUs. It is demonstrated through case study 5 that, if using a file carving technique as an exploratory tool to find many file types from forensic data, utilising OpenCL and all available GPGPU devices would be the best option to employ due to its ability to handle greater amounts of search targets with little loss of performance. However, if file carving for a few specific file types, a multi-threaded CPU technique would be arguably better suited to perform the file carving.

Nonetheless, future work planned to evolve the GPGPU solution presented in the case studies may yield greater processing power from GPGPU devices. Improvements such as hosting more than one processing thread on each available GPGPU device – increasing the level of asynchronous processing – may result in further performance enhancements.

5.1.4 Research question Q4

The introduction of the PFAC multi-string search algorithm in case study 3 demonstrated the best performance improvements across all case studies when conducting string searching for 19 and 40 patterns. When compared to the CPU's modified BM algorithm – the same algorithm as employed by Foremost – and the GPGPU's brute force algorithm, all processing techniques tested demonstrated less depreciation of the time required to conduct searching on larger amounts of search patterns when employing the PFAC algorithm. While PFAC was chosen due to its optimisation for highly-parallelized application; it was seen that the single CPU processing technique also benefitted from notable performance improvements through utilising this algorithm.

This performance enhancement was likewise seen in case study 5, where the PFAC algorithm was used to conduct string searching as part of a file carving operation. Comparing the single CPU processing performance of OpenForensics to that of Foremost, it was observed that the PFAC algorithm employed by OpenForensics could conduct the file carving quicker than the modified BM algorithm used by Foremost. The significance of this comparison indicating that the delivered framework searches data more efficiently than the methods used by Foremost.

While the employment of a GPU paired with a naïve search algorithm showed improvements over current search methods in case study 1, it was found in case study 3 that the employment of modern parallel-friendly algorithms notably contributed to the final performance enhancements achieved. This observation presents compelling

evidence that suggests that the modified BM algorithm could no longer fulfil the requirement of DF investigation due to its inefficient speed when searching for multiple patterns. In its place, this research supports the adoption of parallel optimised multi-string searching algorithms – such as the PFAC algorithm – as a modern standard for future DF tools which rely on string searching to perform analysis.

5.1.5 Research question Q5

Whilst employing multi-threaded CPU and multi-GPU processing with the PFAC algorithm has shown remarkable processing improvements, the performance of conducting file carving has been suggested to be limited by the data transfer speed of the storage device. Case study 4 demonstrated that the method used to read the forensic data from the storage device remains a vital factor which needs to be considered to achieve the best performance from a storage device. The method used to read data was evidenced in this case study to be as important a factor as what processing technique or algorithm was used.

The case studies included in this thesis conducted tests on an SSD and SSD arrays. While SSD and SSD arrays are considerably quicker than traditional HDDs, technological trends have evidenced that SSDs will continue to become much more prevalent in consumer computers (Pal and Memon 2009, p. 59–71). Alongside this technological evolution, it is reasonable to expect that the corpora of digital storage devices seized as part of a DF investigation will evolve alongside it.

Results produced by Foremost to conduct file carving on test platform B, which employed the slowest SSD device on test, showed that Foremost utilised only 29% of the available data transfer performance of the storage device. Contrastingly, utilising a multi-threaded PFAC CPU method in OpenForensics, which was based on the solution presented in case study 4, achieved 97% of the available storage device performance performing file carving.

This suggests that performance conducting file carving in modern DF investigation is no longer bound by storage device performance, but rather a combination of processing performance and storage device performance combined. Foremost evidence that, without modern processing techniques or algorithms, DF tools can struggle to utilise the full performance of a storage device to conduct string searching. Nonetheless, it is important to note that the sequential read performance of the storage device remains to be an insurmountable processing rate limit regardless of the available processing technique used.

5.1.6 Answering the research aim

The results from case study 5 demonstrate that the proposed string searching framework used in case study 4 had notable improvements in speeding up file carving performance— one of the fundamental DF techniques used to analyse forensic evidence. Evidencing that the application of GPGPU technologies and modern parallelisable string searching algorithms reduced the time required to perform file carving in DF investigations**.**

Fundamentally, as DF operations such as file carving rarely deal with searching for single strings when conducting a search, employing a multi-string algorithm is seen by the author as a necessity for all investigatory tools of this field. While this study utilised the PFAC algorithm to conduct string searching, mainly due to its massively-parallel optimisations, it is assumed that other parallelable multi-string search algorithms would reap similar performance benefits over modified single-string searching algorithms when employed to conduct string searching in DF.

The commonplace belief in the DF community stipulates that the speed of data analysis within DF investigation is solely limited by the storage device that forensic data is read from, however, currently used tools such as Foremost were found to use only a fraction of the available performance of the storage device in the tests performed. The

findings of this research present the argument that the speed of data analysis is not limited solely by the storage device transfer rate, but rather a combination of the aforementioned and processing performance— comprising of processing hardware, string searching algorithms used, and the methods employed to read forensic data from the storage device.

This conclusion was further evidenced in trials conducted by Police Scotland with an early prototype tool. Where there was noteworthy evidence that the methods and techniques proposed by case study 2 of this research significantly improve performance over currently employed tools. Suggesting that later developments of this research would again produce more notable improvements.

In concluding remarks, to mitigate the diminishing ability to analyse the increasing amounts of data seized as part of a DF investigation, it is deemed vital to modernising processing methods used by DF tools. It is evidenced from this research that processing improvements could be achieved through many avenues; whether through employing more optimised algorithms or utilising the processing power of GPGPU hardware. It is believed that only through proper tool optimisation and modernisation of processing techniques will we return to the golden age the DF community once enjoyed.

5.2 Future work

5.2.1 OpenForensics future work

This thesis has presented an embodiment of research which largely covers some aspects relating to string searching within DF techniques. Short term aims beyond this thesis are to develop GPGPU processing to include the further application of asynchronous processing. It is envisioned to investigate the benefits of applying the methods and techniques contained within this research to other research areas, such as aiding live network and malware analysis.

With the current trends in technology improving IGP architecture in CPUs, it is intended to continue research into whether the use of these GPGPU processors could lessen the computational strain of other processing tasks associated with DF investigation. As part of this further research, an investigation is planned to measure the possible benefits of applying different algorithms to DF. Utilising hashing algorithms has been evidenced in existing research as being a natural fit for GPGPU processing. Therefore, it seems a logical step to investigate the benefits of applying Bloom filters to identify the presence of file structures within data. Measuring any resulting performance and accuracy differences to the string searching algorithms predominately used in DF tools today.

OpenForensics, the software created as part of this research, will be continued to be developed further with the intent to release to the public. It is planned to continue research into further improving upon the levels of parallelism employed by each GPU in a bid to increase performance file carving for fewer file types. It is also intended to review and integrate advanced file detection methods and fragmented file detection utilising OpenCL and GPU hardware to validate and verify the integrity of the file before reconstruction. Research is also planned to integrate smarter analysis based on the file

system structure detected on the storage device, allowing investigators to target unused space on the file system instead of performing analysis on the full drive.

A comparative file carving review of OpenForensics and a wide selection of commercial and freeware file carving tools is planned to fully explore the benefits of the presented processing framework. It is anticipated that a study from this will be published alongside a paper outlining the processing framework implemented. It is anticipated that the comparative review will highlight other areas where GPGPU processing could aid the DF analysis process.

It is anticipated that with further involvement of forensic divisions and academics will ensure that OpenForensics, and associated researched processing techniques alike, will continue to evolve and benefit the DF community. It is hoped that the content of this research would ignite further investigation and inspiration to evolve a new generation of forensic tools.

5.2.2 Broader applications

An investigation will be done investigating how the OpenForensics processing framework could benefit other fields when employed in other similar applications. Applying the existing processing framework to perform network analysis and live network forensics is believed to be a good fit, as the data analysed and methods used to analyse network traffic share a lot of similarities with DF forensic data analysis. Research is already underway by the author in analysing how the same techniques could be applied to analyse data within live network traffic. It is hoped to further expand this research to include filtering data of forensic importance in network monitoring systems.

This author will also investigate other applications of the processing framework outside of computer forensic fields, as it is apparent that the framework developed could be applied to computational problems of other fields. Initially, an investigation would aim to seek possibilities of applying the processing model to; machine learning,

AI, malware analysis, and health sciences. It is strongly believed that exploring and applying the processing framework to the computational problems of these fields would aid develop and broaden the capabilities of the processing framework.

The author further plans an investigation into visualisation techniques to aid the rapid analysis of live data drawn from DF and network analysis. Whilst this research has proposed a framework to accelerate the acquisition of evidence from unstructured streams of data, the author proposes that powerful visualisation tools could further aid the interpretation of the obtained evidence. Effective visualisation techniques paired with the processing framework would serve as a powerful triage tool in DF investigation. For live network analysis, effective visualisation tools would enhance the ability to quickly identify anomalies in live network traffic.

REFERENCES

Acharya, R 2014, Platform Independent PFAC Implementations using OpenCL on Heterogeneous Parallel Computing, *International Journal of Computer Applications*, 106, (10), pp. 21–24.

Advanced Micro Devices n.d., AMD A-Series Desktop APUs, viewed 19 May, 2016, <http://www.amd.com/en-gb/products/processors/desktop/a-series-apu>.

Agarwal, C, Rasool, A and Khare, N 2013, PFAC Implementation Issues and their Solutions on GPGPU ' s using OpenCL, *International Journal of Computer Applications*, 72, (7), pp. 52–58.

Aho, A V and Corasick, MJ 1975, Efficient String Matching : An Aid to Bibliographic Search, *Communications of the ACM*, 18, (6), pp. 333–340.

Altheide, C and Carvey, H 2011, Digital Forensics with Open Source Tools, 1st ed, Syngress, Waltham.

Arudchutha, S, Nishanthy, T and Ragel, RG 2013, String matching with multicore CPUs: Performing better with the Aho-Corasick algorithm, *2013 IEEE 8th International Conference on Industrial and Information Systems, ICIIS 2013 - Conference Proceedings*, pp. 231–236.

Ayers, D 2009, A second generation computer forensic analysis system, *Digital Investigation*, 6, Elsevier Ltd, pp. S34–S42.

Beek, C 2011, Introduction to File Carving, *McAfee White Paper*.

Bellekens, X, Atkinson, RC, Renfrew, C and Kirkham, T 2013, Investigation of GPU-based Pattern Matching, *The 14th Annual Post Graduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNet2013) (PGNet2013)*, p. 5.

Bellekens, X, Tachtatzis, C, Atkinson, RC, Renfrew, C and Kirkham, T 2014, GLoP: Enabling massively parallel incident response through GPU log processing, *Proceedings of the 7th International Conference on Security of Information and Networks - SIN '14*, (SEPTEMBER 2014), pp. 295–301.

Bhamare, GK and Banait, PSS 2014, Parallelization of Multipattern Matching on GPU, *International Journal of Electronics, Communication & Soft Computing Science and Engineering*, 3, (3), pp. 24–28.

Boyer, RS and Moore, JS 1977, A Fast String Searching Algorithm, *Communications of the ACM*, 20, (10), pp. 762–772.

Breß, S, Kiltz, S and Schäler, M 2013, Forensics on GPU Coprocessing in Databases -- Research Challenges, First Experiments, and Countermeasures, *Proceedings of the 1st Workshop on Databases in Biometrics, Forensics and Security Applications*, pp. 115–129, viewed 17 May, 2013, <http://www.btw-2013.de/proceedings/Forensics on GPU Coprocessing in Databases Research Challenges First Experiments and Countermeasures.pdf>.

Carrier, B, Casey, E and Venema, W 2006, DFRWS 2006 File Carving Challenge, viewed 5 August, 2013,

<http://www.dfrws.org/2006/challenge/>.

Casey, E 2011, Digital Investigation and Computer Crime, 3rd ed, Academic Press.

Charras, C and and Lecroq, T 2004, Handbook of Exact String Matching Algorithms, Kings College Publications.

Chen, C and Wu, F 2013, An Efficient Acceleration of Digital Fonensics Search Using GPGPU, *International Conference on Security and Management*, pp. 1–5.

Chief Police Officers 2008, Good Practice Guide for Computer-Based Electronic Evidence, *Director*, 67, (5), p. 72.

Collange, S, Daumas, M, Dandass, YS and Defour, D 2009, Using graphics processors for parallelizing hash-based data carving, *Proceedings of the 42nd Annual Hawaii International Conference on System Sciences, HICSS*, pp. 1–10.

Crown 2008, Computer Misuse Act 1990, *Computer Misuse Act 1990*, p. 16.

CrystalMark n.d., CrystalDiskMark, viewed 2 June, 2016, <http://crystalmark.info/software/CrystalDiskMark/index-e.html>.

Dharmapurikar, S, Krishnamurthy, P, Sproull, T and Lockwood, J 2003, Deep packet inspection using parallel Bloom filters, *IEEE Micro*, pp. 52–61.

EaseUS n.d., EaseUS Data Recovery Wizard, viewed 23 May, 2016, <http://www.easeus.com/datarecoverywizard/free-data-recovery-software.htm>.

Fang, J, Varbanescu, AL and Sips, H 2011, A Comprehensive Performance Comparison of CUDA and OpenCL, *2011 International Conference on Parallel Processing*, Ieee, pp. 216–225.

File Recovery Ltd. n.d., Undelete 360, viewed 23 May, 2016, <http://www.undelete360.com/>.

Garfinkel, SL 2007, Carving contiguous and fragmented files with fast object validation, *Digital Investigation*, 4, (SUPPL.), pp. 2–12.

Garfinkel, SL 2010, Digital forensics research: The next 10 years, *Digital Investigation*, 7, pp. S64–S73.

Gharaee, H 2014, A Survey of Pattern Matching Algorithm in Intrusion Detection System, *International Symposium on Telecommunications*, Tehran, pp. 946–953.

Grenier, C n.d., PhotoRec, viewed 23 May, 2016, <http://www.cgsecurity.org/wiki/PhotoRec>.

Hales, G 2016, Assisting Digital Forensic Analysis via Exploratory Information Visualisation, Abertay University.

Haseeb, S 2013, Serial and Parallel Bayesian Spam Filtering using Aho- Corasick and PFAC, *International Journal of Computer Applications*, 74, (17), pp. 9–14.

Hybrid DSP n.d., CUDAfy .NET, viewed 9 August, 2013, <http://www.hybriddsp.com/Products/CUDAfyNET.aspx>.

Intel n.d., Intel® Core™ i3-6300 Processor Specifications, viewed 26 May, 2016a, <http://ark.intel.com/products/90731/Intel-Core-i3-6300-Processor-4M-Cache-3_80-GHz>.

Intel n.d., Intel® Core™ i7-5960X Processor Extreme Edition Specifications, viewed 26 May, 2016b, <http://ark.intel.com/products/82930/Intel-Core-i7-5960X-Processor-Extreme-Edition-20M-Cache-up-to-3_50-GHz>.

Intel n.d., Intel® Hyper-Threading Technology, viewed 26 May, 2016c, <http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html>.

Intel n.d., Iris™ Graphics and Intel® HD Graphics Technology, viewed 19 May, 2016d, <https://www-ssl.intel.com/content/www/us/en/architecture-and-technology/hd-graphics/hd-graphics-developer.html>.

International Organization for Standardization n.d., ISO/IEC 17025:2005 - General requirements for the competence of testing and calibration laboratories, viewed 19 May, 2016, <http://www.iso.org/iso/catalogue_detail.htm?csnumber=39883>.

Jeong, Y, Lee, M, Nam, D, Kim, J-S and Hwang, S 2014, High Performance Parallelization of Boyer-Moore Algorithm on Many-Core Accelerators, *2014 International Conference on Cloud and Autonomic Computing*, pp. 265–272.

Karimi, K, Dickson, NG and Hamze, F 2010, A Performance Comparison of CUDA and OpenCL, *ArXiv e-prints*, 1005.2581, p. 12.

Karp, RM and Rabin, MO 1987, Efficient randomized pattern-matching algorithms, *IBM Journal of Research and Development*, 31, (2), pp. 249–260.

Kendall, K, Kornblum, J and Mikus, N n.d., Foremost, viewed 19 May, 2016, <http://foremost.sourceforge.net/>.

Khan, S, Gani, A, Wahab, AWA, Shiraz, M and Ahmad, I 2016, Network forensics: Review, taxonomy, and open challenges, *Journal of Network and Computer Applications*, 66, (March), Elsevier, pp. 214–235.

Khronos Group n.d., OpenCL, viewed 6 August, 2013, <http://www.khronos.org/opencl/>.

Kouzinopoulos, CS, Assael, J-AM, Pyrgiotis, TK and Margaritis, KG 2015, A Hybrid Parallel Implementation of the Aho-Corasick and Wu-Manber Algorithms Using NVIDIA CUDA and MPI Evaluated on a Biological Sequence Database, *International Journal on Artificial Intelligence Tools*, 24, (1).

Lin, C-H, Liu, C-H, Chien, L-S and Chang, S-C 2013, Accelerating Pattern Matching Using a Novel Parallel Algorithm on GPUs, *IEEE Transactions on Computers*, 62, (10), pp. 1906–1916.

Lin, C, Liu, C and Chang, S 2011, Accelerating Regular Expression Matching Using Hierarchical Parallel Machines on GPU, *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*, pp. 1–5.

Lin, CH, Tsai, SY, Liu, CH, Chang, SC and Shyu, JM 2010, Accelerating string matching using multi-threaded algorithm on GPU, *GLOBECOM - IEEE Global Telecommunications Conference*, pp. 1–5.

Marziale, L, Richard, GG and Roussev, V 2007, Massive threading: Using GPUs to increase the performance of digital forensics tools, *Digital Investigation*, 4, pp. 73–81.

Merola, A 2008, Data Carving Concepts, *Sans Institute*, 11, p. 40.

Microsoft n.d., Introduction to the C# Language and the .NET Framework, viewed 24 May, 2016, <https://msdn.microsoft.com/en-gb/library/z1zx9t92.aspx>.

Mohan, D 2010, Faster file matching using GPGPUs, University of Delaware.

Mohr, A n.d., Quantum Computing in Complexity Theory and Theory of Computation.

Mokaram, NE 2015, New Pattern Matching Approaches Comparison.

Nance, K, Hay, B and Bishop, M 2009, Digital forensics: defining a research agenda, *System Sciences, 2009. HICSS'09.*, pp. 1–6.

Nugent, H 1995, Prosecuting computer crimes, *International Review of Law, Computers*, pp. 159–182, viewed 19 May, 2016, <https://www.justice.gov/sites/default/files/criminal-ccips/legacy/2015/01/14/ccmanual.pdf>.

Nvidia n.d., GeForce GTX 980 Specifications, viewed 26 May, 2016, <http://www.geforce.co.uk/hardware/desktop-gpus/geforce-gtx-980/specifications>.

Pal, A and Memon, N 2009, The evolution of file carving, *IEEE Signal Processing Magazine*, 26, (2), pp. 59–71.

Piriform n.d., Recuva, viewed 23 May, 2016, <http://www.piriform.com/recuva>.

Pungila, C and Negru, V 2012, A highly-efficient memory-compression approach for GPU-accelerated virus signature matching, *Information Security - ISC 2012, Springer, LNCS*, 7483, pp. 354–369.

Raghavan, S 2013, Digital forensic research: current state of the art, *CSI Transactions on ICT*, 1, (1), pp. 91–114.

Rasool, A and Khare, N 2013, Generalized Parallelization of String Matching Algorithms on SIMD Architecture, *International Journal of Computer Science and Information Security*, 11, (5), pp. 6–16.

Richard III, GG and Roussev, V 2006, Digital Forensic Tools: The Next Generation, *Digital crime and forensic science in cyberspace*, pp. 76–91.

Richard III, GG and Roussev, V 2005, Scalpel: A Frugal, High Performance File Carver, *Proceedings of the 2005 Digital Forensics Research Workshop (DFRWS '05)*, pp. 1–10.

Roussev, V and Richard III, GG 2004, Breaking the Performance Wall: The Case for Distributed Digital Forensics, *Digital Forensics Research Workshop*, (September), pp. 1–16.

Scientific Working Group on Digital Evidence 2002, Best Practices for Computer Forensics, viewed 19 May,

2016, <https://www.swgde.org/documents/Archived Documents/2004-11-15 SWGDE Best Practices for Computer Forensics v1.0>.

Skrbina, N and Stojanovski, T 2012, Using parallel processing for file carving, *ArXiv e-prints*, 1205.0103, (March).

Sommer, P 1999, Intrusion detection systems as evidence, *Computer Networks*, 31, (23–24), pp. 2477–2487.

Soroushnia, S, Daneshtalab, M, Plosila, J and Liljeberg, P 2014, Heterogeneous Parallelization of Aho-Corasick Algorithm, *Practical Applications of Computational Biology & Bioinformatics*, pp. 153–160.

Soroushnia, S, Daneshtalab, M, Plosila, J and Pahikkala, T 2014, High Performance Pattern Matching on Heterogeneous Platform Related works, *Journal of Integrative Bioinformatics*, 11, (3), pp. 253–264.

Takahashi, R and Inoue, U 2012, Parallel Text Matching Using GPGPU, *2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pp. 242–246.

Thambawita, DRVLB, Ragel, R and Elkaduwe, D 2014, To use or not to use: Graphics processing units (GPUs) for pattern matching algorithms, *Information and Automation for Sustainability*, pp. 1–4.

Tran, N-P, Lee, M, Hong, S and Bae, J 2013, Performance Optimization of Aho-Corasick Algorithm on a GPU, *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 1143–1152.

Tran, N-P, Lee, M, Hong, S and Shin, M 2012, Memory efficient parallelization for Aho-Corasick algorithm on a GPU, *Proceedings of the 14th IEEE International Conference on High Performance Computing and Communications, HPCC-2012 - 9th IEEE International Conference on Embedded Software and Systems, ICESS-2012*, Ieee, pp. 432–438.

Vasiliadis, G, Antonatos, S, Polychronakis, M, Markatos, E and Ioannidis, S 2008, Gnort: High performance network intrusion detection using graphics processors, *Recent Advances in Intrusion Detection*, pp. 116–134.

Wagner, B 2009, Deep Packet Inspection and Internet Censorship: International Convergence on an 'Integrated Technology of Control', *Global Voices*.

WiseCleaner n.d., Wise Data Recovery, viewed 23 May, 2016, <http://www.wisecleaner.com/wise-data-recovery.html>.

Wu, P 2013, CLgrep: A Parallel String Matching Tool.

Zha, X and Sahni, S 2011, Fast in-Place File Carving for Digital Forensics, *Forensics in Telecommunications, Information, and Multimedia*, pp. 141–158.

Zha, X and Sahni, S 2013, GPU-to-GPU and Host-to-Host Multipattern String Matching on a GPU, *IEEE*

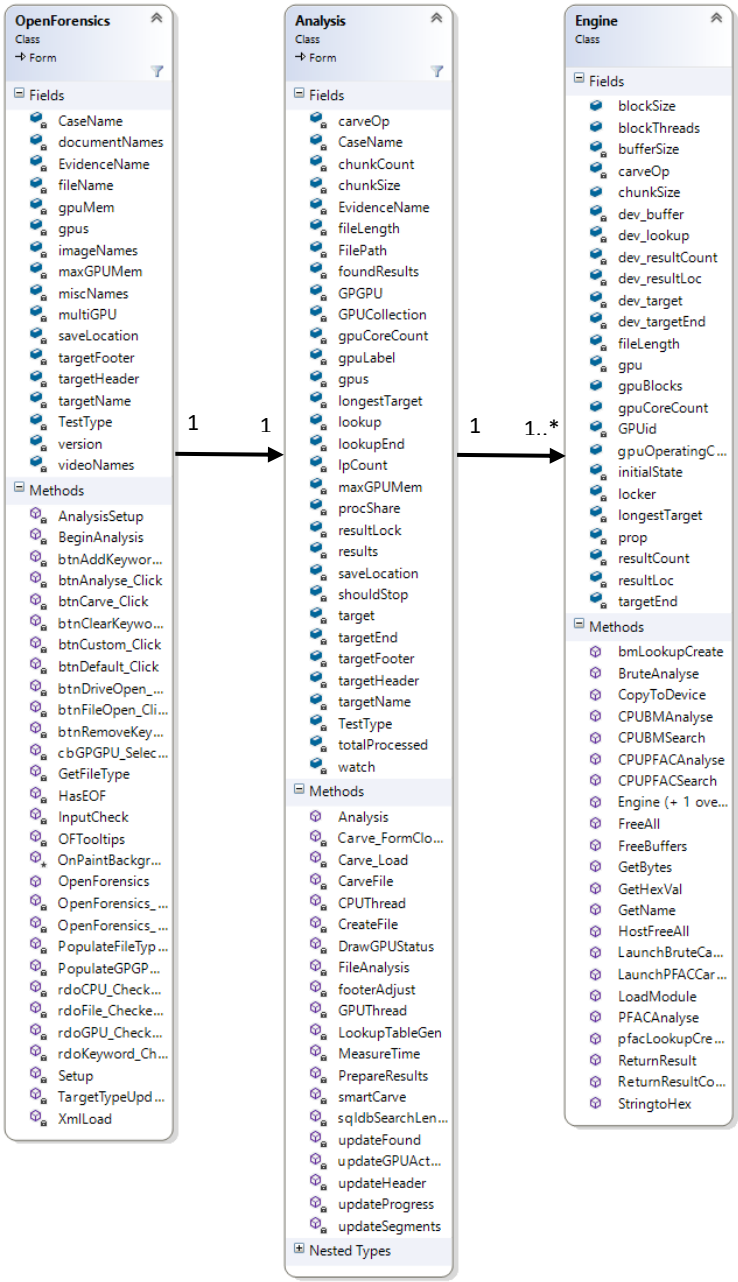
Transactions on Computers, 62, (6), pp. 1156–1169.

Zha, X and Sahni, S 2011, Multipattern String Matching On A GPU, *IEEE Symposium on Computers and Communications*, pp. 277–282.

Appendices

Appendix A

OPENFORENSICS CLASS DIAGRAM



Appendix B

FOREMOST CONFIGURATION FILES

B.1 Foremost String Searching Configuration File Settings – 5 String Search

jpg	y	1000	\xFF\xD8\xFF\xE0\x00\x10
jpg	y	1000	\xFF\xD8\xFF\xE1\x35\xFE
gif	y	1000	\x47\x49\x46\x38\x39\x61
gif	y	1000	\x47\x49\x46\x38\x37\x61
png	y	1000	\x89\x50\x4E\x47\x0D\x0A\x1A\x0A

B.2 Foremost String Searching Configuration File Settings – 19 String Search

jpg	y	1000	\xFF\xD8\xFF\xE0\x00\x10
jpg	y	1000	\xFF\xD8\xFF\xE1\x35\xFE
gif	y	1000	\x47\x49\x46\x38\x39\x61
gif	y	1000	\x47\x49\x46\x38\x37\x61
png	y	1000	\x89\x50\x4E\x47\x0D\x0A\x1A\x0A
tiff	y	1000	\x49\x49\x2A\x00
tiff	y	1000	\x4D\x4D\x00\x2A
mpg	y	1000	\x00\x00\x01\xBA
mpg	y	1000	\x00\x00\x01\xB3
wmv	y	1000	\x30\x26\xB2\x75\x8E\x66\xCF\x11\xA6\xD9\x00\xAA\x00\x62\xCE\x6C
wma	y	1000	\x30\x26\xB2\x75
doc	y	1000	\xD0\xCF\x11\xE0\xA1\xB1
docx	y	1000	\x50\x4B\x03\x04\x14\x00\x06\x00
pdf	y	1000	\x25\x50\x44\x46
zip	y	1000	\x50\x4B\x03\x04
zip	y	1000	\x50\x4B\x05\x06
zip	y	1000	\x50\x4B\x07\x08
rar	y	1000	\x52\x61\x72\x21\x1A\x07\x00
rar	y	1000	\x52\x61\x72\x21\x1A\x07\x01\x00

B.3 Foremost String Searching Configuration File Settings – 40 String Search

```
jpg     y       1000    \xFF\xD8\xFF\xE0\x00\x10
jpg     y       1000    \xFF\xD8\xFF\xE1\x35\xFE
gif     y       1000    \x47\x49\x46\x38\x39\x61
gif     y       1000    \x47\x49\x46\x38\x37\x61
png     y       1000    \x89\x50\x4E\x47\x0D\x0A\x1A\x0A
tiff    y       1000    \x49\x49\x2A\x00
tiff    y       1000    \x4D\x4D\x00\x2A
wim     y       1000    \x4D\x53\x57\x49\x4D
mpg     y       1000    \x00\x00\x01\xBA
mpg     y       1000    \x00\x00\x01\xB3
mp4     y       1000    \x00\x00\x00\x14\x66\x74\x79\x70\x69\x73\x6F\x6D
mp4     y       1000    \x00\x00\x00\x18\x66\x74\x79\x70\x33\x67\x70\x35
mp4     y       1000
        \x00\x00\x00\x1C\x66\x74\x79\x70\x4D\x53\x4E\x56\x01\x29\x00\x46\x4D\x53\x4E\x56\x6D\x70\x34\x32
mov     y       1000    \x00\x00\x00\x14\x66\x74\x79\x70\x71\x74\x20\x20
m4v     y       1000    \x00\x00\x00\x18\x66\x74\x79\x70\x6D\x70\x34\x32
wmv     y       1000    \x30\x26\xB2\x75\x8E\x66\xCF\x11\xA6\xD9\x00\xAA\x00\x62\xCE\x6C
mkv     y       1000    \x1A\x45\xDF\xA3\x93\x42\x82\x88\x6D\x61\x74\x72\x6F\x73\x6B\x61
wma     y       1000    \x30\x26\xB2\x75
m4a     y       1000    \x00\x00\x00\x20\x66\x74\x79\x70\x4D\x34\x41\x20
doc     y       1000    \xD0\xCF\x11\xE0\xA1\xB1
docx    y       1000    \x50\x4B\x03\x04\x14\x00\x06\x00
pdf     y       1000    \x25\x50\x44\x46
zip     y       1000    \x50\x4B\x03\x04
zip     y       1000    \x50\x4B\x05\x06
zip     y       1000    \x50\x4B\x07\x08
zip     y       1000    \x50\x4B\x03\x04\x14\x00\x01\x00\x63\x00\x00\x00\x00\x00
rar     y       1000    \x52\x61\x72\x21\x1A\x07\x00
rar     y       1000    \x52\x61\x72\x21\x1A\x07\x01\x00
xar     y       1000    \x78\x61\x72\x21
xz      y       1000    \xFD\x37\x7A\x58\x5A\x00
jar     y       1000    \x4A\x41\x52\x43\x53\x00
jar     y       1000    \x5F\x27\xA8\x89
iso     y       1000    \x43\x44\x30\x30\x31
cso     y       1000    \x43\x49\x53\x4F
img     y       1000    \x50\x49\x43\x54\x00\x08
img     y       1000    \x51\x46\x49\xFB
img     y       1000    \x53\x43\x4D\x49
cas     y       1000    \x5F\x43\x41\x53\x45\x5F
```

```
rpm     y       1000    \xED\xAB\xEE\xDB
mof     y       1000    \xFF\xFE\x23\x00\x6C\x00\x69\x00\x6E\x00\x65\x00\x20\x00\x31\x00
```

B.4 Foremost File Carving Configuration File Settings – 9 File Types

```
jpg     y       10485760  \xFF\xD8\xFF\xE0\x00\x10        \xFF\xD9
jpg     y       10485760  \xFF\xD8\xFF\xE1\x35\xFE        \xFF\xD9
gif     y       10485760  \x47\x49\x46\x38\x39\x61        \x00\x3B
gif     y       10485760  \x47\x49\x46\x38\x37\x61        \x00\x3B
png     y       10485760  \x89\x50\x4E\x47\x0D\x0A\x1A\x0A          \x49\x45\x4E\x44\xAE\x42\x60\x82
mpg     y       10485760  \x00\x00\x01\xBA     \x00\x00\x01\xB7
mpg     y       10485760  \x00\x00\x01\xB3     \x00\x00\x01\xB7
docx    y       10485760  \x50\x4B\x03\x04\x14\x00\x06\x00          \x50\x4B\x05\x06
pdf     y       10485760  \x25\x50\x44\x46     \x0A\x25\x25\x45\x4F\x46
```

Appendix C

POLICE SCOTLAND CORRESPONDANCE

From: [redacted]
Sent: 08 November 2015 21:46
To: Bayne, Ethan
Cc: Ferguson, Ian
Subject: Re: File Carving

Hi Ethan -

Sorry I didn't get a chance to get back to you sooner, but I am barely ever in the office these days (or so it seems) and haven't therefore been working on too many cases.

Having said that, I made sure I unleashed your file carver on a couple of test cases (ie, rigged drives!) to see what it revealed. I was pleased to find that it managed to recover files from a 'live' drive very quickly indeed and found absolutely everything I was after, all of which is of course positive and what I hope you were after. On average I found it worked about 160% faster than an equivalent product, which is all the more remarkable given the obvious shortcoming to the product.

The tool I compared it with is smart enough to know what is an unallocated cluster and what is not, and searched only those sectors, whereas your product seemed to search the entire drive content from one end to the other. That it managed to do this faster than a different tool which was far more selective (and in doing so searched only about 50% of the 120Gb drive's storage area). The drawback really is that your program assumes that everything is a drive 'artefact' and not a live file, which means that we cannot differentiate between one and the other. Finding deleted and unallocated files is a substantial - and slow - part of what we do, and anything that can speed that up would be of benefit. Unfortunately we would have to have some way of saying that what a program recovers definitely comes from only that unallocated region, and not from the actual filing system. Am I making sense?

Having said that - it's really good!

kind regards

[redacted]